Enforcing Security on Autonomous Vehicle Searches Through the Quantification of Opacity

By Bryony H. Schonewille

A thesis submitted to the Graduate Program in Electrical And Computer Engineering in conformity with the requirements for the Degree of Master's of Applied Science

Queen's University
Kingston, Ontario, Canada
December, 2021

Abstract

Recently, topics of security have been explored in the field of discrete-event systems (DES). By modelling these systems with DES, the evolution of the processes can be captured, allowing for different vulnerabilities to be noticed. The DES field also provides a different set of tools which can generate new strategies to tackle the security problems in these systems. The motivating problem that this work focuses on is a group of autonomous vehicles traversing some terrain while trying to cooperatively complete a task such as searching for a target. This work shows that strategies can be employed that remove the need for communication under certain conditions. When no such strategy can be developed, it can be useful to have a tool to classify a system's security. In DES, this has traditionally been captured by the notion of *opacity*. Unfortunately, this framework has many limitations. Opacity can only be applied to systems with a specific secret and it cannot tell you how obscured a system is. To address these failings, this work develops the concept of *degree of opacity*. This framework can measure the degree to which a system is secure based on a supplied criterion. It also can be applied to systems as a whole eliminating the dependency on a specific secret. Degree of opacity is a more general version of opacity and can be handled in similar ways. Degree of opacity can be enforced using supervisory control to produce a minimally restrictive supervisor. Alternatively, what

is communicated to a vehicle or observed by a vehicle can be controlled, rather than restricting the vehicle's movements. Unfortunately, non-monotonicity of observability means that it cannot be easily used to enforce degree of opacity like controllability was. To combat this, a strategy called *transition pairing* is developed which allows monotonicity to be achieved. This strategy is applied to construct an algorithm to pick which communications can be communicated to achieve a target degree of opacity.

Acknowledgements

This research was undertaken at Queen's University, which is situated on traditional Anishinaabe and Haudenosaunee territory.

I would like to thank my supervisor, Dr. Karen Rudie, for all of the support and guidance she has provided over the course of this Master's. From advice, to help with idea generation, to lots of edits, this thesis would not have been possible without her help.

The support provided from General Dynamics Land Systems–Canada and Defence Research and Development Canada in the joint research project was invaluable. The collaboration of different groups in this project provided unique perspectives and inspiration, without which this work would not have been possible.

Additionally I would like to acknowledge my colleagues in the Queen's Discrete-Event Systems group. All of them are amazing people and listeners, always providing helpful feedback on work. I would especially like to thank Richard H. Moulton, Jeremy Kulchyk, and Richard Ean. Collaborating with them on joint work was an incredible experience which greatly contributed to this thesis.

Finally I would like to thank my partner, Kyle Laporte. His care and support was irreplaceable. Thank you for reminding me that caring for myself is important and results in better work.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

With the increasing development of technology and artificial intelligence, industries are constantly asking how these new advancements can be applied to improve the processes in their sectors. While technology can drastically increase productivity, worker safety, and consistency in a work flow, it can also add new risks or challenges. Just as technology has developed to be used as tools, so too has it been developed for nefarious purposes. In particular, programmable technology and communication tools, while being extremely useful and versatile, are major security risks. Data can be protected with methods like encryption, but in some cases, this may not be able to be applied due to costs or it may not provide enough security for an important task. This thesis will demonstrate how the current notion of security in the field of discrete-event systems, opacity, can be expanded upon to provide a greater area of applications with a more refined and adjustable notion of security.

1.2 Motivation

This project was funded in part by General Dynamics Land Systems and Defence Research and Development Canada. These organizations, like many others, are interested in how autonomous vehicles can be used to cooperatively complete tasks. However, cooperation is most easily facilitated by communication, which can lead to system insecurities. This was my motivation to explore how the field of discrete-event systems could be used to design tools and strategies that could be employed to enhance the security of systems of autonomous vehicles. Throughout the work, I used the sample problem of a maze search as an example of how the developed tools and methods could be applied to systems of autonomous vehicles. While these tools are motivated by systems of autonomous vehicles, they are by no means restricted to these applications. In fact these tools help to expand the reach of the field of discrete-event systems with respect to security.

1.3 Thesis Objectives

The objective of this work is to develop strategies that will diminish or eliminate security risks in cooperative terrain searches. While methods like cryptography are widely used to secure communications, they can fail and they may not be able to be applied in some cases due to lack of time and resources. Developing strategies that introduce targeted confusion for the adversary will prevent the adversarial observer from determining a system's secret information. Our goal is to extend the current notion of opacity, an indicator of whether secret information can be discerned by an adversary. We hope to develop an extension which is a more general measure that also captures to what degree secret information is obscured.

1.4 Thesis Contributions

This thesis contributes new concepts to the aspect of security in the field of discrete-event systems and new strategies for secure cooperative terrain searches. It proffers a movement policy for a pair of cooperating vehicles searching a terrain that eliminates the need for communication in cases with certain properties. This thesis also presents a probabilistic examination of how this movement policy could be extended to three or four vehicles. The main contribution is a novel security framework for discrete-event systems called *degree of opacity* that extends the concept of *opacity* by measuring the degree to which an adversary is confused. Two different algorithms are presented that enforce degree of opacity using two unique techniques. Additionally, a transition pairing technique is presented which generates monotonic behaviour for observable transitions.

1.5 Thesis Outline

The remainder of this thesis is organized as follows. Chapter 2 will provide an overview of discrete-event systems concepts that are necessary for the understanding of this work. Chapter 3 will explore the motivating industry problem and how strategies can be developed to eliminate the need for communication between agents. In Chapter 4 we introduce *degree of opacity* as a measure of traditional opacity and explore ways that it can be enforced. This work is concluded in Chapter 5 and possible future directions are discussed.

Chapter 2

Background

2.1 Introduction to Discrete-Event Systems

In this chapter we will overview the fundamental concepts that this work has drawn upon from the field of discrete-event systems. This information can be found with more details and description in [1, 2].

A deterministic finite automaton (DFA), called a *system* or *plant* G, is described as a five-tuple:

$$G = (Q, \Sigma, Q_0, \delta_G, Q_m)$$

The set Q is the set of *states* that the system could be in. The set Σ is the set of possible *events* that could trigger the system to move from state to state. The set of *initial states* is denoted Q_0, where $Q_0 \subseteq Q$ and these are the possible states where the system will begin its evolution. For the purpose of this thesis, there will only be one initial state, turning the set Q_0 into a singleton initial state q_0. The set Q_m is the set of *marked states* in the system and $Q_m \subseteq Q$. Marked states are states of some importance to the system, either desirable states or a system configuration to avoid.

For some problems this may be unnecessary and thus omitted, leaving the system to be described by the four-tuple $G = (Q, \Sigma, Q_0, \delta_G)$. The transition function, δ_G, describes the transitions that connect the states in the system. This partial function is $\delta_G : Q \times \Sigma \to Q$ and a when a transition of event $\sigma \in \Sigma$ from state $q \in Q$ is defined, it is denoted $\delta(q, \sigma)!$.

Figure 2.1 contains a simple deterministic finite automaton. For this system, the state set is $Q = \{1, 2, 3, 4\}$. The initial state, denoted by the one input arrow, is $q_0 = 1$. The marked state is denoted by a second ring around the state. In this system, $Q_m = \{4\}$. The event set consists of all of the transition labels so $\Sigma = \{a, b, c\}$. Finally the transition function, δ_G, defines the five transitions in the system.

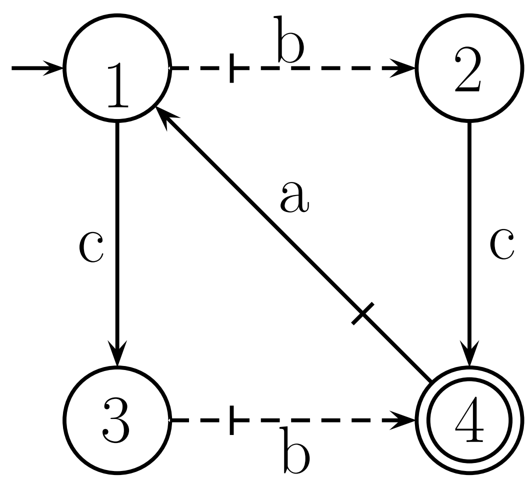


Figure 2.1: Example of a deterministic finite automaton

We can define a function that maps a state to a set of events that are valid outgoing transitions from this state.

Definition (Active Event Function and Active Event Set)**.** *Given state q, the active event set $\Gamma : Q \to 2^{\Sigma}$ is mathematically defined as follows.*

$$\Gamma(q) = \{\sigma \in \Sigma \mid \delta(q, \sigma)!\}$$

For example, in Figure 2.1, we see that $\Gamma(1) = \{b, c\}$.

Events in the event set Σ can be concatenated together to form strings. The set of strings formed by an event set Σ is denoted by Σ^*. This set is also called the *Kleene closure* of the event set. The transition function δ_G can be extended recursively to act on $Q \times \Sigma^*$. For a string $s \in \Sigma^*$, event $\sigma \in \Sigma$, and state $q \in Q$, this recursion can be seen in the following statement: $\delta(q, s\sigma) = \delta(\delta(q, s), \sigma)$. The empty string is denoted by ε. The set of strings in Σ^* that are valid paths in a system G is called the *language* of the system, $L(G)$.

The language for the system shown in Figure 2.1 is

$$L(G) = \{\varepsilon, c, b, bc, cb, bca, cba, bcac, ...\}$$

Each string can also be associated with a set of prefixes. For a given string $t \in \Sigma^*$, the set of prefixes is denoted $\bar{t}$ and defined as $\bar{t} = \{s \in \Sigma^* | \exists s' \in \Sigma \text{ and } ss' = t\}$.

The set of events can be partitioned to reflect different properties of these events. One of these properties is controllability. Events that can be chosen to be enabled or disabled are called *controllable events*. They make up the set $\Sigma_c \subseteq \Sigma$. Events that cannot be enabled or disabled are called *uncontrollable events*, Σ_{uc}, and $\Sigma = \Sigma_c \dot{\cup} \Sigma_{uc}$. In an automaton diagram, controllable events are marked with a tick across the transition. For the example in Figure 2.1 events a and b have this tick mark and thus form the set of controllable events.

Control actions can be enforced by an agent that can interact with a system. This agent is called a supervisor. A *supervisor* can be represented as a set of enable and disable decisions, with these decisions sometimes written in a binary form. A supervisor can also be represented as a deterministic finite automaton, S. This automaton will have the same event set as the original system with a potentially modified state space and transition function. This new system, S, represents the behaviour of the original system, G, that the supervisor will enable, e.g., if there is a defined transition in S, that transition will be enabled in G. The complete closed-loop behaviour of the supervisor enacting control on the plant is denoted S/G [3].

Consider the example in Figure 2.1. If a supervisor, S, enables event b at both states in G where it can occur and disables event a at state 4 in G, then the expected behaviour will be automaton S/G in Figure 2.2.

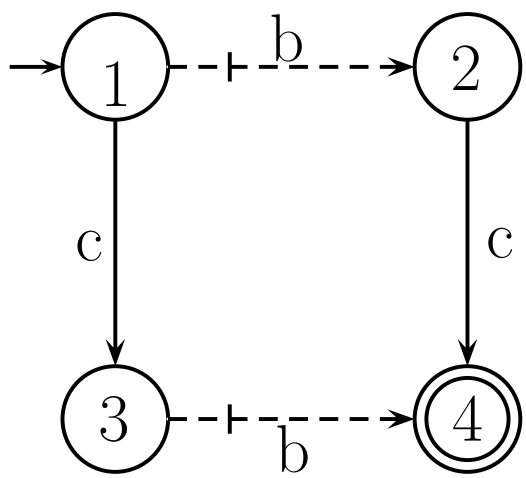


Figure 2.2: Behaviour of system in Figure 2.1 when a supervisor enables all occurrences of event b and disables event a

Another property that can partition the event set is observability. There is the set of observable events, Σ_o, and the set of unobservable events, Σ_{uo}, and $\Sigma = \Sigma_o \dot{\cup} \Sigma_{uo}$. Events that are unobservable cannot be seen when they occur, so a supervisor may not be certain which state the system is currently in. Unobservable events are marked with a dashed line in diagrams and this can be seen in Figure 2.1. The only unobservable event in this system is event b.

When analyzing a system from an observer's point of view, it is useful to be able to separate out only the observable events. This can be done using a *natural projection* of the observable event set to project out the string of observable events.

Definition (Projection of Observable Events)**.** *Given an alphabet* Σ *and observable alphabet* $\Sigma_o \subseteq \Sigma$*, the projection* $P_{\Sigma_o} : \Sigma^* \to \Sigma_o^*$ *is defined recursively below:*

$$P_{\Sigma_o}(\varepsilon) = \varepsilon$$

$$P_{\Sigma_o}(\sigma) = \begin{cases} \sigma \ \textit{if}\ \sigma \in \Sigma_o \\ \varepsilon \ \textit{otherwise} \end{cases}$$

$$P_{\Sigma_o}(s\sigma) = P_{\Sigma_o}(s)P_{\Sigma_o}(\sigma) \ s \in \Sigma^*, \sigma \in \Sigma.$$

When there is a set of unobservable events, it can be very useful to analyze a system using an observer automaton. An observer of a system depicts the observable behaviour of the system. The state set for the observer is groups of states from the original system that an observer would estimate the current state of the system to be in based on observed events. One traditional way of constructing an observer is to replace all events in Σ_{uo} with the empty string, ε. After all unobservable events are replaced, the system becomes a *nondeterministic finite automaton* (NFA). The observer can be generated by converting this system back to a DFA. The constructed observer of a system, G, is denoted as follows:

$$H = \mathrm{Obs}_{\Sigma_o}(G) = (\mathcal{A}, \Sigma_o, A_0, \delta_H, \mathcal{A}_m)$$

Going back to the running example in this section, Figure 2.3 displays the observer of the system in Figure 2.1. It demonstrates that since event b is unobservable, an observer cannot be certain which specific state the system currently resides. An observer estimates that the system is either in one of states 1 or 2 initially and then in one of states 3 or 4 after the observer sees a c. When an a is observed after the c, once again the observer will not know if the plant is in state 1 or state 2.

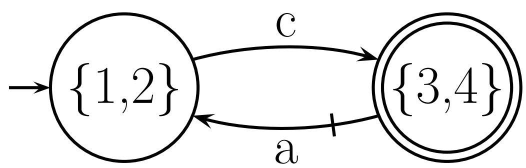


Figure 2.3: Observer of the system in Figure 2.1

Another useful object in analyzing partially observable systems is the *parallel composition* of the system G with its observer H [2]. It is denoted M and defined below.

$$\begin{aligned} M = G \mid\mid H &= Ac(Q \times \mathcal{A}, \Sigma \cup \Sigma_o, q_0 \times A_0, \delta_M, Q_m \times A_m) \\ &= (X, \Sigma, x_0, \delta_M, X_m) \end{aligned}$$

In the above equation, $Ac()$, is a function which returns the accessible part of the automaton and δ_M is the parallel composition's transition function which is as follows. For $\sigma \in \Sigma$, for $q \in Q$, $A \in \mathcal{A}$,

$$\delta_M((q, A), \sigma) = \begin{cases} (\delta_G(q, \sigma), \delta_H(A, \sigma)) & \text{if } \delta_G(q, \sigma)! \text{ and } \sigma \in \Sigma_o \\ (\delta_G(q, \sigma), A) & \text{if } \delta_G(q, \sigma)! \text{ and } \sigma \notin \Sigma_o \\ \text{undefined} & \text{otherwise} \end{cases}$$

This structure will allow us to track which states an observer *thinks* the plant is in when the plant is in some particular state. Each state, x, in a parallel composition will be depicted in diagrams as a fraction where $x = \frac{q}{A}$. The numerator of the state, q, is the true state of the system. The denominator, A, is an observer's estimate of states where the system could be at this point given the observable information.

The system in Figure 2.1 can be combined with its observer in Figure 2.3 to produce the parallel composition in Figure 2.4.

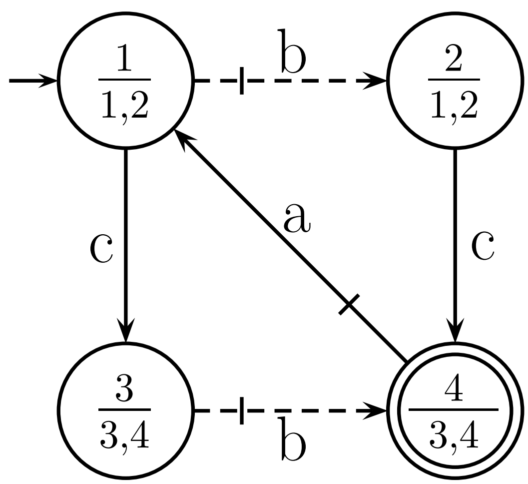


Figure 2.4: Parallel composition of the system in Figure 2.1 and observer in Figure 2.3

2.2 Opacity

In discrete event systems, if security is an issue, then opacity is frequently the subject of discussion. Opacity was originally a concept in computer science that captured whether an observer could unambiguously determine the current state or a given property, like a secret, about the system. This was then applied to discrete event systems by Saboori and Hadjicostis [4]. There are many different versions of opacity that have been presented in the literature, but Saboori and Hadjicostis' initial definition was *current state opacity*, also known as (S, P)-opacity. It says that given a system G with a set of secret states $S \in Q$ is opaque if an observer cannot determine unambiguously whether the system has reached any of the secret states.

Definition (Current-State Opacity (CSO))**.** *Given a DFA $G = (Q, \Sigma, \delta_G, q_0)$ and the projection P_{Σ_o} for a set of observable events $\Sigma_o \subseteq \Sigma$, G is opaque with respect to a set of secret states $S \subseteq Q$ under the projection P_{Σ_o} if*

$$\begin{aligned} &\forall\ t \in L(G), \delta_G(q_0, t) \in S \implies \\ &\quad \exists s \in L(G) \setminus \{t\} \mid (\delta_G(q_0, s) \notin S)(P_{\Sigma_o}(s) = P_{\Sigma_o}(t)). \end{aligned}$$

Other versions of opacity include strong and weak opacity, language based opacity, and k-step opacity [5,6]. Only current-state opacity is defined in this thesis since many versions of opacity have been proven to be equivalent [7].

Chapter 3

Security of Maze Searches

3.1 Motivating Industry Application

This thesis was funded through a project in collaboration with General Dynamics Land Systems. As such, this work was heavily influenced by applications of interest to the project. This section will discuss these influences and the specific applications that will be targeted by the work in this thesis.

As society has become more technologically advanced, people have been avidly experimenting with ways to integrate new technology into various physical systems and processes. The study and design of autonomous worker agents to perform tasks has garnered a lot of interest. One variety of autonomous agent that is of interest to the project this thesis was funded by is the intelligent autonomous vehicle. This type of vehicle is being experimented with and implemented in a large variety of applications. These include mining operations, underwater exploration, and terrain traversal. The application that was focused on was terrestrial traversal, in particular the cooperation of a collection of autonomous vehicles in a terrestrial environment to accomplish a predetermined task. These agents could be used to perform a variety of tasks. These include resource distribution, terrain mapping, search and rescue, and area maintenance.

This thesis will specifically be focused on search and rescue applications and the formal set-up for the base problem is described in Problem 1. The system will involve a group of agents that are autonomous vehicles. Modelling of the cooperation of a group of autonomous vehicles has only been approached by DES a few times [8, 9]. An agent is *autonomous* if it can execute a set of directives without having to be directly controlled by someone.

Problem 1 (Cooperative Search and Rescue Problem)**.** *Given a group of autonomous vehicles and known terrain with a target of unknown location, devise a policy for the vehicles to locate and convene at the location of target while prioritizing search efficiency and maintaining a system secure from adversaries.*

Adversaries are observers of the system whose goals are to also locate the target and/or the agents. In this set-up, they are passive, they do not have any capabilities of affecting any communications or the agents themselves.

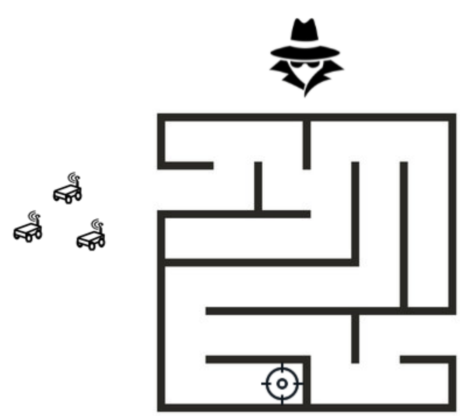


Figure 3.1: Depiction of Problem 1

Figure 3.1 shows an artistic depiction of the problem presented in Problem 1. In this image, three communicating vehicles are searching a maze for the marked target while an adversary is monitoring the global communications.

The situation introduced in Problem 1 can have many variations.

For more complex adaptations of Problem 1 it may be useful for the agents to take a more active role by using some level of intelligence. For the purpose of this thesis, an agent is *intelligent* if has the ability to make its own decisions based on the information it has available.

3.2 Communication-Free Cooperation of Two Vehicles

The concepts presented in this section were either presented in Kulchyk et al. [10], which I co-authored, or built upon concepts in said paper. The specific notation and presentation style of the material differs from its original presentation in Kulchyk et al. [10] to better fit the notation and flow of this thesis. My specific contributions will be outlined further in this section.

When attempting to design a solution to a problem similar to the one outlined in Problem 1, one immediately apparent tool to use is communication. Communication will allow for the vehicles to describe what they experience to each other. This allows for the search to be efficient, each agent searching areas they know the others are not. However, as the motivating problem describes, it is very possible that there are external "observers" who can listen in on these wireless communications. If these observers have nefarious intentions, then the communications that were useful may quickly become harmful, potentially giving the adversary enough information to reach the target first. Since the communication is useful for the search efficiency, it becomes a balancing act, boiling down to the question, "How much can be communicated to allow for as efficient a search as possible while still allowing for the system to remain secure?"

Surprisingly, for some cases the answer to this question is no communication at all. The goal of the communication in the first place is to allow for maximal efficiency of the maze search. If a movement policy can be designed for the agents such that communication is unnecessary, both goals of being secure and efficient are achieved. Our work [10] demonstrates how this can be done with a version of Problem 1.

The problem that we focused on is described in Problem 2. This problem statement was developed jointly by the authors.

Problem 2 (Search of Terrain without Loops by a Pair of Autonomous Vehicles)**.** *Given a pair of autonomous vehicles, unknown terrain which is a system of bidirectional transitions without loops and has event set $\Sigma = \{N, E, S, W\}$, and a target of unknown location, devise a pair of movement policies for the vehicles to locate the location of target with maximal efficiency without communication.*

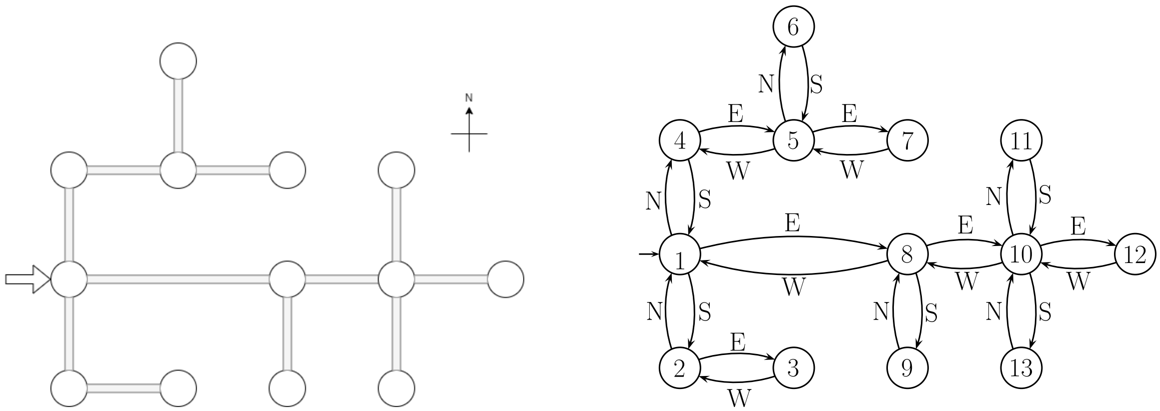
An example of the terrain that the above problem statement describes is a series of paths that branch from an initial location. The no loop requirement means that two branches never connect, more paths only occur from branching of existing paths.

These paths are all bidirectional, meaning that they can be traversed forwards and backwards. In an automaton, these can be represented by a pair of directed edges with *inverse events*.

Definition (Inverse Events)**.** *Inverse events are a pair of events, σ and σ', such that $\delta(\delta(q, \sigma), \sigma') = q$ and $\delta(\delta(q, \sigma'), \sigma) = q$ for any $q \in Q$. If σ' is an inverse event of σ, then:*

$$\sigma' = \sigma^{-1}$$

Since the system is assumed to have the event set $\Sigma = \{N, E, S, W\}$, the event set can be naturally broken into two pairs of inverse events. Intuitively, these pairs are $\{N, S\}$ and $\{E, W\}$. Figure 3.2a depicts a path system and Figure 3.2b is the result of translating the path system into an automaton.



(a) Example path system for Problem 2 (b) Automaton for path system in Figure 3.2a

Figure 3.2: Example showing how a system fitting the criteria for Problem 2 can be translated into an automaton

Before any movement policies can be synthesized to produce an efficient terrain search, terrain search efficiency needs to be defined. To introduce the structures needed to capture efficiency, the concept of "path pruning" is necessary. When a path is pruned, it removes any movements from the vehicle's path that will not change the end destination, producing the shortest path to the destination state.

Definition (Path Pruning Projection)**.** *The path pruning projection is a function which removes any "back and forth" behaviour from a string of events. This is defined below in a recursive manner. For any* $s, u, v \in \Sigma^*$

$$
\begin{aligned}
P_{prn} &: L(G) \rightarrow L(G) \\
P_{prn}(\epsilon) &:= \epsilon \\
P_{prn}(suv) &:= \begin{cases} P_{prn}(sv) & \textit{if } u = \sigma_1\sigma_2...\sigma_n\sigma_n^{-1}...\sigma_1^{-1} \textit{ for some } \sigma_1, \sigma_2, ..., \sigma_n \in \Sigma \\ suv & \textit{otherwise} \end{cases}
\end{aligned}
$$

What this definition says is for any string, if there exists a substring $u \in \Sigma^*$ where $u = \sigma_1\sigma_2...\sigma_n\sigma_n^{-1}...\sigma_1^{-1}$ for some $\sigma_1, \sigma_2, ..., \sigma_n \in \Sigma$, then this string can be pruned by removing u from the string. This process is repeated until there are no more substrings of the above form. Consider the system in Figure 3.2b. Suppose there is a vehicle which traversed the path $SEWNESNE$, ending up in State 10. What is this vehicle's pruned path?

$$
\begin{aligned}
P_{prn}(SEWNESNE) &= P_{prn}(SNESNE) \text{ since } E = W^{-1} \\
&= P_{prn}(ESNE) \text{ since } S = N^{-1} \\
&= P_{prn}(EE) \text{ since } S = N^{-1} \\
&= EE
\end{aligned}
$$

If we begin from the left to search for substrings of the form described above, the first one that is found is EW, the second and third events. As per the projection, this substring is removed and the projection is calculated on this new string. Removing

EW from the string has caused the first two events, SN, to form a substring which has the appropriate form. This substring is removed from the string and the projection is recalculated. The resultant path's second and third events form another appropriate substring, SN. After this substring is removed, the pruned path is EE. No more substrings of the correct form can be found in this string, so this is the vehicle's pruned path. In this example, the substrings were found by searching through the string, starting from the left side. While this method was used, it does not matter which substring is removed first if there are multiple substrings present in the current string. For example, in the first step, the substring SN formed by the sixth and seventh events could be removed first leading to $SEWNEE$. If you continue the process, you have $P_{prn}(SEWNEE) = P_{prn}(SNEE) = P_{prn}(EE) = EE$. The order of the removal of the substrings is irrelevant because the process will continue until all such substrings have been removed and the removal of one substring has no effect on another substring except to potentially allow events that were blocked by the removed substring to become a part of another substring.

For maze searches, efficiency will be defined with respect to any dead-ends in the system. For this problem set-up, efficiency will not be determined by the number of dead-ends that are searched. If the terrain is unknown, or the location of the target is unknown, it is not reasonable to expect a search policy to avoid searching any dead-ends since the vehicles cannot know ahead of time which paths are the dead-ends. Since searching some dead-ends is unavoidable, it is desirable to make sure that any states that have been searched aren't searched again. Once one vehicle has verified a state to not have the target, there is no need for it or any other vehicle to search that state again unless it leads directly to the target. This concept will be the basis for the terrain search efficiency definition. This is a natural way to define search efficiency for this setup because if the vehicles had unrestricted communication, they could communicate their exact paths, allowing for the other vehicles to avoid the states the other vehicles have searched. Dead-end states are any states that are not on the pruned path from the initial state to the marked state.

Definition (Set of Dead-End States)**.** *Let the set of dead-end states be Q_D. This set is defined as*

$$Q_D = \{q \in Q | \nexists s \in \bar{t} \textit{ where } \delta(q_0, s) = q \textit{ given } \delta(q_0, t) = q_m, P_{prn}(t) = t\}$$

As defined in Problem 2 the vehicles are searching for a target state, q_m. This is their only objective so a vehicles does not need to continue traversing the maze once it finds q_m, as the search will be terminated and the vehicle will stop moving. Since the vehicle will stop moving at this point, the set of all states that it visits in the search is the set of states that it has passed through on its path to q_m. This is explicitly defined as the *set of visited states* below.

Definition (Set of Visited States)**.** *The set of visited states for vehicle V_i is denoted $Q(V_i)$. This is the set of states that the vehicle has entered by the time it finds state q_m. Let $t \in \Sigma^*$ be the sequence of events generated by V_i as it goes to q_m (i.e., $\delta(q_0, t) = q_m$). The set of visited states is defined as*

$$Q(V_i) = \{q \ | \exists s \in \bar{t} \textit{ where } \delta(q_0, s) = q\}$$

Definition (Terrain Search Efficiency)**.** *A terrain search is maximally efficient if no dead-end states are searched by more than one agent. For a set of vehicles indexed* *with the variable i, a search is maximally efficient if*

$$Q_D \cap \left(\bigcap_i Q(V_i) \right) = \emptyset$$

With the search efficiency defined, the policies to solve Problem 2 can be described. The type of policy that will be used is called a *priority policy*. The priority policy is a decision-making process that makes decisions on possible events at a state based on a given *priority function*.

Definition (Priority Policy)**.** *A priority function, $\mathcal{P}$, is a bijective function $\mathcal{P} : \Sigma \rightarrow \{1, 2, ..., |\Sigma|\}$ which assigns a unique priority to each event in the event set of a system. The event with highest priority is the event with the lowest rating, 1.*

Definition (Priority Policy)**.** *A priority policy, $\pi_{\mathcal{P}}$, is a decision-making process that selects the highest priority event of available events given a priority function, $\mathcal{P}$. Given a subset of the event set, $\Sigma' \subseteq \Sigma$, an event is returned based on the following function.*

$$\pi_{\mathcal{P}}(\Sigma') = \arg\min_{\sigma \in \Sigma'} \mathcal{P}(\sigma)$$

A priority policy can be summarized into a priority tuple where the position of an event in the tuple is its priority. Each priority policy has a complementary policy called a *reverse policy*.

Definition (Reverse Priority Policy)**.** *A reverse priority policy for the priority policy $\pi_{\mathcal{P}}$ is $\pi_{R(\mathcal{P})}$. It is a priority policy using the the reverse priority function of $\mathcal{P}$, $R(\mathcal{P})$. It is defined on a given event $\sigma \in \Sigma$ as follows.*

$$R(\mathcal{P})(\sigma) = |\Sigma| - \mathcal{P}(\sigma) - 1$$

This function generates a priority which is the reverse of the priorities assigned by $\mathcal{P}$.

In the situation described in Problem 2, the event set is the cardinal directions, $\Sigma = \{N, E, S, W\}$. With this set, an example priority function $\mathcal{P}$, would be $\mathcal{P}(E) = 1$, $\mathcal{P}(S) = 2$, $\mathcal{P}(N) = 3$, and $\mathcal{P}(W) = 4$. As a priority tuple this policy would be $\mathcal{P} = (E, S, N, W)$. This priority function as a vehicle's priority policy is reflected by the following description: "go East first if able, then South if able, then North if able, then West". The reverse priority function for the example priority function above is $\mathcal{P} = (W, N, S, E)$.

These policies define how the vehicles will navigate the terrain, but to be a solution to Problem 2, it needs to be shown that they will produce a maximally efficient search.

Theorem 1 (Maximal Efficiency of a Priority Policy and its Reverse Policy in a Simple Maze Search)**.** *Given two autonomous vehicles,* V_1 *and* V_2 *searching for a target as outlined by Problem 2, using priority policy* $\pi_{\mathcal{P}}$ *and its reverse, the search is maximally efficient as defined in Definition 3.2.*

Since G has only one initial state, both vehicles enter the system from the same location. Also G is a system without loops so each state in the system has a unique shortest path from the initial state and all paths from the initial state that end in a given state can be pruned to the shortest path. This implies that the agents will always enter any given state q the first time using the same transition.

The vehicles' movement policies are designed so that when they enter a state, upon deciding which event to take, they will search that entire sub-tree before returning to the initial state if the marked state is not found. Knowing this behaviour, it is enough to analyze the states on the path to the marked state. If it can be guaranteed that the vehicles only go down the same path if it leads to the target state, then this will guarantee that if a vehicle searches a dead-end sub-tree of the system, the other vehicle will not search that sub-tree since it will encounter the target before backtracking to that dead-end. This will be shown by keeping track of the decisions that each vehicle makes at a state using the following set.

Definition (Set of Event Decisions of a State)**.** *The set of event decisions of a vehicle, V_i, for a given state $q \in Q$ is denoted $\Sigma(V_i, q)$. It is defined as follows:*

$$\Sigma(V_i, q) = \{\sigma | \delta(q, \sigma)! \wedge \delta(q, \sigma) \in Q(V_i)\}$$

This set represents the set of events that the vehicle V_i has used to leave state q by the time it finds q_m.

Proof of Theorem 1. Consider state $q \in Q_p = Q \backslash Q_D$ which has the active event set $\Gamma(q) \subseteq \Sigma$. Since G has no loops, there is a unique event, $\sigma \in \Sigma$, to enter q for the first time. Remove σ^{-1} from the active event set since the vehicles will never back track up the tree since all above paths lead to dead end states. Next, order the events according to their priority based on V_1's policy, $\pi_{\mathcal{P}}$. Name this tuple $\Gamma'(q)$. Since $q \in Q_p$, $\exists s \in \Sigma* \ |\delta(q, \sigma' P_p(s)) = q_m$ for only $\sigma' \in \Gamma'(q)$. Let $I(\sigma') = i$, this is the index of σ with respect to the tuple $\Gamma'(q)$.

Each vehicle will pick events based on their policy. If the event is not σ', it will lead to a dead-end sub-tree, causing the vehicle to return to state q. Once a vehicle chooses event σ', it will never return to state q because it will find q_m and terminate the search first. Which events will each vehicle select before it chooses σ'?

Since the events in $\Gamma'(q)$, are ordered by V_1's policy, $\pi_{\mathcal{P}}$,

$$\Sigma(V_1, q) = \{\sigma'' \mid I(\sigma'') \leq i\}$$

Conversely, since V_2 uses the policy $\pi_{R(\mathcal{P})}$,

$$\Sigma(V_2, q) = \{\sigma'' \mid I(\sigma'') \geq i\}$$

This means that

$$\Sigma(V_1, q) \cap \Sigma(V_2, q) = \{\sigma\}$$

This result tells us that the decisions the vehicles make for any $q \in Q_D$ are irrelevant. This is the case because no shared event leads to a dead-end state so dead-ends will only be searched by one vehicle or the other. This implies that

$$Q(V_1) \cap Q(V_2) \subseteq Q \backslash Q_D$$

The search efficiency can then be calculated.

$$Q_D \cap Q(V_1) \cap Q(V_2) \subseteq Q_D \cap Q \backslash Q_D = \emptyset$$

This means that the search with these policies is maximally efficient. □

This proof demonstrates how the use of a priority policy and its reverse priority policy leads to a maximally efficient search. Since this is shown without any communication between the vehicles, communication is unnecessary and this is indeed a solution to Problem 2.

Theorem 1 appears as the main result in Kulchyk et al., [10, Theorem 12]. The proof presented in Kulchyk et al. [10] is an alternate proof to my proof above. The proof in Kulchyk et al. [10] is a concise proof developed by Jeremy Kulchyk with inspiration drawn from a longer proof I had created. My proof of Theorem 1 is a refined version of my original longer proof.

Since the priority policy and its pair have been proven to offer maximal efficiency, let it now be demonstrated with an example. Figure 3.3 shows an the result of a maze search simulation using the reverse priority policy technique. White blocks denote the boundaries of this maze. The initial state is coloured purple while the target is coloured blue. Two vehicles are searching this system for the target. The first vehicle is using the priority policy, $\pi_{\mathcal{P}} = (E, W, N, S)$, and the second vehicle is using the reverse policy, $\pi_{R(\mathcal{P})} = (S, N, W, E)$. Any state that is only searched by vehicle 1 is coloured red, any state only searched by vehicle 2 is coloured yellow, and states searched by both are coloured green. Finally, any unsearched states are coloured black. It can be clearly seen that the only states that are coloured green are states that are on the direct path to the blue marked state. This means that no dead-end states are searched by both agents, as proved with Theorem 1.

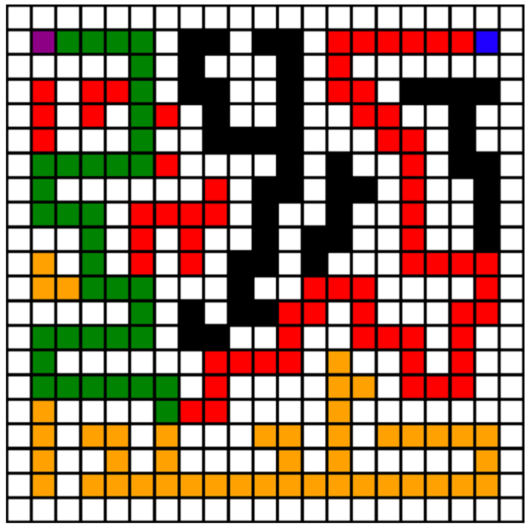


Figure 3.3: Example of vehicles with reverse priority policies searching a maze

3.3 Communication-Free Cooperation of Three and Four Vehicles

The set-up presented in Problem 2 is restrictive in certain ways. One way it is restrictive is the need for a system without loops and bidirectional paths. This extension has been explored further in Kulchyk's thesis [11]. The other main restriction is the number of vehicles used in the search. Allowing for more than two vehicles to participate in a search would be useful. More vehicles, if used appropriately, will help to speed up the search by covering more ground. If more vehicles are added to the search, the solution as it stands cannot be directly applied since priority policies form reverse policy pairs. This means that there is no third policy that is reverse to the pair of reverse policies used by the other two vehicles.

Even though there is no third reverse policy, the concept can still be extended to groups of more than two agents. With no third reverse policy, preserving maximum search efficiency cannot be guaranteed in a communication-free setting. Since maximum search efficiency cannot be achieved under the constraints of the problem, good candidates for a third policy can be determined heuristically by analyzing the spread of the search. The search spread is a useful measure since this will increase the speed of the search, even if maximum efficiency cannot be guaranteed.

Given a priority policy and its reverse, which priority policy is a good candidate for the third vehicle to promote the largest search coverage? Clearly, the worst case is to repeat either of the policies assigned to the first two vehicles. Since the vehicles are using their policies to determine their paths, repeating a policy would have two vehicles searching the exact same states, rendering one vehicle irrelevant. Since this is the worst case, to find the priority policy that will produce the most divergence in search areas, we will need to find a policy that is as different as it can be from both other policies. Since the policies of the first two vehicles are a reverse policy pair, differentiating a policy from one of the two policies will make it more similar to its reverse policy. This turns the search for the third policy into a balancing act of divergence from each of the other policies. To determine the best candidate for a third policy, start by analyzing the decisions made by the two policies at every intersection type. In a system where $\Sigma = \{N, E, S, W\}$, there are eleven different intersections types. These can be broken down to one four-way junction type, $\{N, E, S, W\}$, four three-way junction types, $\{N, E, W\}, \{N, E, S\}, \{N, W, S\}, \{E, W, S\}$, and six two-way junction types, $\{N, E\}, \{N, W\}, \{N, S\}, \{E, W\}, \{E, S\}, \{W, S\}$. These eleven intersection types are the only types of situations where the vehicles will have to make a movement decision based on the problem set-up. Table 3.1 summarizes how many times each event of a given priority is chosen among the eleven different intersection types. The percentage chosen column reflects the percentage of decisions that will result in that policy given an even distribution of intersection types. The distribution

Priority	Times Chosen	Percentage Chosen
1	7	$\approx 64\%$
2	3	$\approx 27\%$
3	1	$\approx 9\%$
4	0	0%

Table 3.1: Distribution of decisions between the different events based on their priorities and $\{N, E, S, W\}$ intersections

of intersection types will differ from system to system, however the larger the system becomes, the more likely it is that the distribution will become even.

With this information, it is clear if a third policy prioritizes the low priority events of a given policy, it will increase divergence of the third policy with the given policy. Indeed, this is the principle that allows for a reverse priority policy to maximize the search efficiency of two vehicles in the above section. When constructing a third policy, the lower priority events from both initial policies will need to be prioritized. However, since the other two policies are reverses of each other, it is difficult to determine which policy will be most opposite. Consider the initial priority policy, $\pi_{\mathcal{P}} = (N, E, S, W)$, and its reverse $\pi_{R(\mathcal{P})} = (W, S, E, N)$. Which policy will be a good candidate for a third policy, $\pi_{\mathcal{P}'}$. The information in Table 3.2 compares different options for $\pi_{\mathcal{P}'}$, specifically analyzing the number of times the decisions made by these policy candidates overlap with the decisions made by $\pi_{\mathcal{P}}$ and $\pi_{R(\mathcal{P})}$.

In Table 3.2, the decisions that each proposed policy make at each intersection type is recorded. Cells are coloured red if the decision matches that made by $\pi_{\mathcal{P}} = (N, E, S, W)$ and coloured blue if the decision matches the decision made by $\pi_{R(\mathcal{P})} = (W, S, E, N)$. Since $\pi_{\mathcal{P}}$ and $\pi_{R(\mathcal{P})}$ are a reverse priority policy pair, the decisions they

	$\pi_{\mathcal{P}'}$			
Intersection	(E, S, W, N)	(E, W, N, S)	(S, N, W, E)	(S, E, N, W)
$\{N, E, S, W\}$	E	E	S	S
$\{N, E, W\}$	E	E	N	E
$\{N, E, S\}$	E	E	S	S
$\{N, S, W\}$	S	W	S	S
$\{E, S, W\}$	E	E	S	S
$\{N, E\}$	E	E	N	E
$\{N, S\}$	S	N	S	S
$\{N, W\}$	W	W	N	N
$\{E, S\}$	E	E	S	S
$\{E, W\}$	E	E	W	E
$\{S, W\}$	S	W	S	S

Table 3.2: Analysis of decisions made at a given intersection by a proposed third policy and their overlap with policy (N, E, S, W) and reverse policy (W, S, E, N)

make at each intersection are distinct. This means that no cell in the Table 3.2 can be both blue and red. Since we are picking a policy that will be the most different from both $\pi_{\mathcal{P}}$ and $\pi_{R(\mathcal{P})}$, fewer overlaps in decisions is desirable. This table only depicts four different options for a third policy to conserve space. All policies that are not included in this table have more decision overlaps than the policies analyzed in the table, making them worse candidates than the ones shown. Different properties of interest that are derived from Table 3.2 are presented in Table 3.3.

One important thing to note is that all policies that are being considered in Table 3.3 have as close to a 1:1 ratio of decisions between $\pi_{\mathcal{P}}$ and $\pi_{R(\mathcal{P})}$. When the ratio is close or equal to 1:1, this means that if the third vehicle follows either Vehicle

	$\pi_{\mathcal{P}'}$			
Property	(E, S, W, N)	(E, W, N, S)	(S, N, W, E)	(S, E, N, W)
$\pi_{\mathcal{P}}$ Overlap Count	4	4	4	3
$\pi_{R(\mathcal{P})}$ Overlap Count	3	4	4	4
Independent Decision Count	4	3	3	4
$\pi_{\mathcal{P}}$:$\pi_{R(\mathcal{P})}$	4:3	1:1	1:1	3:4
Least Prioritized Event	N	S	E	W

Table 3.3: Summary of policy properties based on the information in Table 3.2

1 or Vehicle 2 down a path, it will have more opportunities for their paths to diverge, increasing the spread of the search. If, for example, a policy had a skewed ratio like 7:1, then if the third vehicle made the same decision as $\pi_{\mathcal{P}}$, it would have a hard time diverging paths from the vehicle with $\pi_{\mathcal{P}}$ since only four intersection types would allow for the paths to diverge (one decision matched with the other vehicle and three non-matched decisions). With a ratio that is 1:1, there would be seven intersection types that would allow for divergence from either $\pi_{\mathcal{P}}$ or $\pi_{R(\mathcal{P})}$ (four matched with the other vehicle and three unmatched decisions).

The first three rows of Table 3.3 are counts of the overlaps of the decisions with $\pi_{\mathcal{P}}$, $\pi_{R(\mathcal{P})}$, or neither. Policies (E, S, W, N) and (S, E, N, W) both have the fewest overlaps in total decisions, each making four independent decisions. This fact makes them good candidates for the third policy. These policies are of course only good candidates as a third policy for the two specific policies the above analysis is performed on. However, these policies can be generalized by referring to the priority of the events instead of the specific events with those priorities. Let $\pi_{\mathcal{P}} = (\sigma_1, \sigma_2, \sigma_3, \sigma_4)$. Now the candidates for $\pi_{\mathcal{P}'}$ can be generalized as:

$$\pi_{\mathcal{P}'} = \begin{cases} (\sigma_2, \sigma_3, \sigma_4, \sigma_1) \\ (\sigma_3, \sigma_2, \sigma_1, \sigma_4) \end{cases}$$

These policies are good third policy candidates if the plan is to only introduce a third vehicle, however what if a fourth vehicle is going to be added to the search as well? As discussed earlier, to increase the spread of the search, the policies should be as different from one another as possible. As shown in Theorem 1, a reverse priority pair is maximally efficient. With the need for four policies, two reverse priority pairs would allow for the greatest differences between all four policies. Unfortunately, if either (E, S, W, N) or (S, E, N, W) are reversed, the reversed policy will have the same first priority as an existing policy. As shown in Table 3.1, the first priority is chosen the most often by a large margin. Having two policies with the same first priority would make these two policies very likely to make the same decisions, decreasing the search divergence. Instead, consider the pair of policies (E, W, N, S) and (S, N, W, E). While they have one fewer independent decision when compared to the other policies presented in Table 3.3, they both have 1:1 decision ratios and together form a reverse priority pair. For these reasons, they would be better options for the policies in a system of four vehicles. As with the candidates for the third policy, this result can also be generalized. Label $\pi_{\mathcal{P}} = (\sigma_1, \sigma_2, \sigma_3, \sigma_4)$, using these priority labels, the candidates for the third and fourth vehicles can be written as follows.

$$\pi_{\mathcal{P}'} = (\sigma_2, \sigma_4, \sigma_1, \sigma_3)$$

$$\pi_{R(\mathcal{P}')} = (\sigma_3, \sigma_1, \sigma_4, \sigma_2)$$

Since the divergence of policies is a determining factor for choosing a third and fourth policy, it is useful to be able to calculate how likely these policies are to diverge and how many intersections it will take for these policies to diverge. Both of these can be determined using probability. For two policies $\pi_{\mathcal{P}}$ and $\pi_{\mathcal{P}'}$, let $div(\pi_{\mathcal{P}}, \pi_{\mathcal{P}'})$ be the probability that their paths will diverge after n intersections where m is the number of overlapping decisions between the two policies. This probability equation is presented in (3.1).

$$div(\pi_{\mathcal{P}}, \pi_{\mathcal{P}'}) = 1 - \left(\frac{m}{11}\right)^n \tag{3.1}$$

This equation is based on the following concept. As a pair of vehicles approach an intersection, there are eleven types of intersections it could be. Of those eleven intersection types, there are m of them that would lead the vehicles to make the same decision. This means that there is an $\frac{m}{11}$ chance that the vehicles' decisions will match. This chance is compounded over the n intersections that the vehicles encounter, each intersection producing a new chance for the vehicles to have their paths diverge. This produces the probability that their paths will be the same, and the divergence probability is obtained by subtracting this probability from one.

This description does make it evident that this calculation relies upon an equal probability of encountering an intersection of any given type. With a large enough system this is a reasonable assumption, although if a system is known to favour a given intersection type more, the "$\frac{m}{11}$" portion of the equation can be adjusted to weight different intersection types based on their likelihood to appear. A generalized equation that implements this idea is shown in (3.2). In this equation, m_i is an indicator function which is one if the ith intersection type has a matched decision and w_i is the percentage, from zero to one, of intersections in the system that are the ith type.

$$div(\pi_{\mathcal{P}}, \pi_{\mathcal{P}'}) = 1 - \left(\sum_{i=1}^{11} m_i w_i \right)^n \tag{3.2}$$

Equation (3.1) can also be rearranged to instead evaluate how many intersections it would take to have two policies diverge with a certain probability. This rearranged equation is shown in (3.3).

$$n = \log_{\frac{m}{11}}(1 - div(\pi_{\mathcal{P}}, \pi_{\mathcal{P}'})) \tag{3.3}$$

Equation 3.3 can be used to analyze the divergence of the policies that were thought to be best for a third or fourth vehicle in the example above. The third or fourth policies all had a maximum of four overlapping decisions with any other vehicle's policy. Say we want a 99% chance that these policies will diverge. Apply Equation (3.3):

$$\begin{aligned} n &= \log_{\frac{4}{11}}(1 - 0.99) \\ &\approx 4.55 \end{aligned} \tag{3.4}$$

Since 4.55 rounds up to 5, the result of (3.4) means that after five intersections, these policies should diverge paths. This is a very good result for large systems. This may seem restrictive for smaller systems, however these systems should not require additional search vehicles. While this section explores the use of three and four vehicles in a search, additional policies can be chosen using the same reasoning as above.

While this is the extent to which this solution extension was explored in this thesis, it would be useful to determine if the efficiency of these policies could be bounded. An efficiency bound would allow for a given terrain system to be analyzed, where the speed of the search would be balanced with the added efficiency.

3.4 Communication-Free Cooperation Potential Extensions

Problem 2 was set up with a specific set of requirements. These requirements allow for the priority policy framework to work with maximal search efficiency. The requirements are restrictive so not many scenarios will fall into this niche. This set-up was expanded in Section 3.3 to include more than two vehicles with implications of how it could be further expanded to more than four vehicles. The other main restriction on the problem set-up is the need for an system with bidirectional paths and no loops. Applications can vary wildly so it would be ideal to extend this work to a larger class of systems. This avenue was pursued in Kulchyk's MASc thesis [11]. This problem could also be modified for different scenarios; e.g., working with known terrain, preplanning paths, modifying paths during the search, changing the system goal, etc.

While this problem can be modified in many different ways, these changes may make communication necessary. For example, what if vehicles need to efficiently convene at the target location? While they may know what movement policy the vehicle that found the target used, they would not be sure what area the vehicle ended up in. In this case, some limited communication may facilitate this goal since the successful vehicle could provide the other agents with some information that would allow them to narrow their search. However, as discussed earlier, any communication is a vulnerability that can be exploited by a nefarious observer. The new goal for this situation is to balance the communication used with the system's security. This balance is explored in depth in the following chapter.

Chapter 4

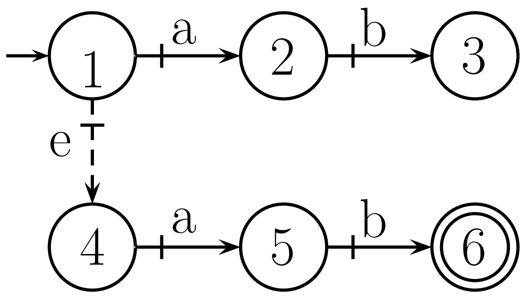
Degree of Opacity

4.1 Opacity in Discrete Event Systems

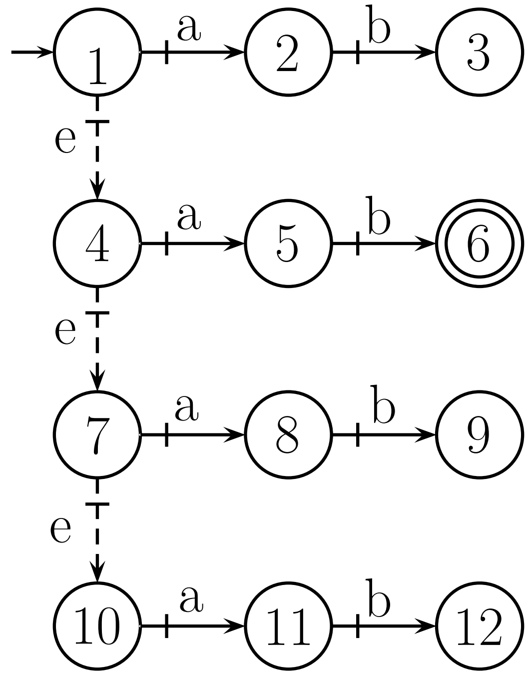
Opacity is a very useful tool to classify whether a system is secure. Many problems benefit from its application however it is also true that not all DES security problems are captured with opacity alone. While opacity can classify whether a system is opaque or not, that is all it offers. In this framework, there is no concept of *how* opaque and, by extension, how secure a given system is. Consider the two systems in Figure 4.1. For this example, as is standard with state-based opacity problems, any secret state in a system will be marked. Both systems have one marked state, State 6. System 1 in Figure 4.1a has only two rows of states while System 2, shown in Figure 4.1b, extends System 1 and has four rows of states.

The observers for both systems in Figure 4.1 are shown in Figure 4.2. Consider the estimate states that include the marked state, State 6. For System 1, the estimate including State 6 includes State 3 as well. For System 2, the estimate including the marked state includes State 3 but also State 9 and State 12.

As discussed in Chapter 2, if we consider a common definition of opacity, CSO

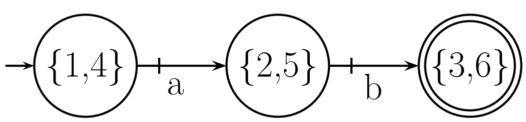


(a) System 1

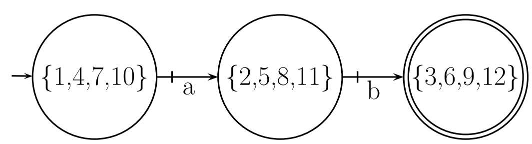


(b) System 2

Figure 4.1: Two systems for comparison of the application of opacity



(a) Observer for system in 4.1a



(b) Observer for system in 4.1b

Figure 4.2: Observers for the systems in 4.1 for the application of opacity comparison

defined in Definition 2.2, a system is opaque if every secret state is indistinguishable from a non-secret state. For both above systems, the estimate with State 6 at least includes State 3, which is a non-secret state. Therefore, by the traditional opacity framework, both of these systems are considered to be opaque. While it is useful to know that the secret states for both systems are protected in some capacity, by looking at Figure 4.2, it is clear that the secret state in System 2 is more obscured than the secret state in System 1 since it has triple the amount of indistinguishable non-secret states. Despite this fact, opacity classifies both systems as opaque and makes no reference to how obscured the secret state is. By adding more rows of states to System 2, more non-secret states would be indistinguishable with the marked state, further driving home the point that two opaque systems do not necessarily offer the same amount of security. For many situations it may not be enough to only be confused by one non-secret state, which is the only guarantee that traditional opacity provides.

Another weak spot for traditional opacity is how it is tied to fixed secrets, either a group of secret states or a secret language. What if there is a system where it is undesirable to have any observer precisely know the current state of the system? In this case, all states can be considered secret. Since all states could be secret so there are no non-secret states to obscure the secret states. As shown in the formal definition of CSO, Definition 2.2, a system's opacity relies upon the presence of secret and non-secret states. With all states having an unknown secret status, CSO cannot be applied. This weak spot is particularly concerning for the motivating industry problem, Problem 1, introduced in Chapter 3 since it falls into this category. To demonstrate this, suppose there was a group of autonomous vehicles searching terrain which can be modelled by the automaton in Figure 4.3. If the locations of these agents need to be obscured or the target location is unknown, there is no specific secret that needs to be obscured since the agents and the target could be in any state. Instead, all states must be obscured since they could be the target location or be traversed by an agent.

Different versions of opacity have been developed to relax its restrictive binary

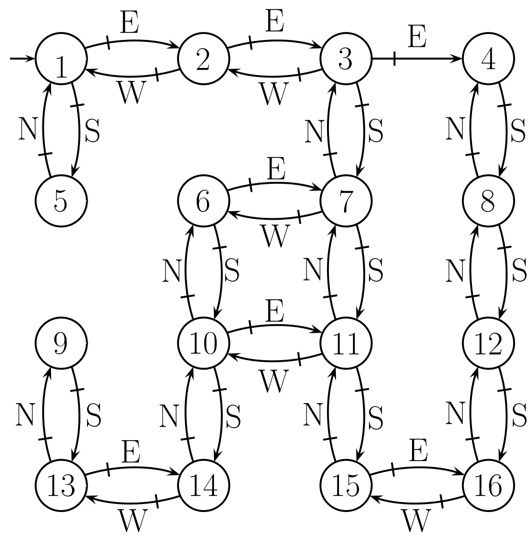


Figure 4.3: Automaton modelling terrain for a vehicle search

quality such as restrictive probabilistic opacity [12], probabilistic current state opacity [13], and k-step opacity [6]. Each of these requires a specific system secret. Additionally they either involve notions of probability, quantifying the chance that the adversary knows the system secret, or they capture how many steps can be taken before the adversary will know that the system was in a secret state. All of these are limited in representing the protection that a secret has based on the system configuration.

In the following section, a new opacity framework will be introduced. It will extend traditional opacity to allow for it to be applicable to problems that fall into the situations mentioned above. Specifically,it will allow for the motivating situation to also be captured by opacity.

4.2 Degree of Opacity

The material in this section was presented in the paper "Enforcing Degree of Opacity with Supervisory Control" [14] from the Conference on Decision and Control 2021. This paper was written by myself and co-authored by Richard Hugh Moulton and Karen Rudie. All of the material in this section was solely my work, with the exception of minor edits and refinement of symbology.

In Section 4.1, a few security problems were presented where opacity is not sufficient to describe them. Unfortunately, the motivating industry problem, Problem 1, involves both of the weaknesses presented. Due to this, it is necessary to develop a different opacity framework so that opacity can be applied. The new framework that I developed is presented in this section and is called *degree of opacity*. Before the definition of DoO can be presented, a useful helper function must be described.

Definition (Opacity Weighting Function)**.** *Given a system $G = (Q, \Sigma, q_0, \delta_G)$, the opacity weighting function for that system is*

$$w : 2^Q \to \mathcal{R}^+ \ ,$$

where w is monotonically increasing with respect to subset inclusion.

The opacity weighting function is an arbitrary function which describes relationships that are important to a given system's security. This function will be used as a tool to determine what added security a given state will add to an estimate. As an example, consider the system in Figure 4.4. Let this system represent a terrain which has three locations. State 1 currently houses a friendly agent and there are two other connected states. State 2 is 10m away from State 1 while State 3 is 500m away. Suppose an adversarial observer is going to traverse the the terrain in attempt to find the agent. If we had to choose either State 2 or State 3 to be indistinguishable with State 1, which state is the better choice? If State 2 is indistinguishable with State 1, if the adversary searches State 2 and discovers the agent is not there, it is a very short distance to discover the agent in State 1. If State 3 is indistinguishable instead, if the adversary searches State 3 first, it would take a lot longer for them to continue on to State 1. That extra traversal time may even allow for the agent to move to a different state and not be caught by the adversary. This clearly depicts how depending on the situation, different states can offer more obscurity to a secret state than others.

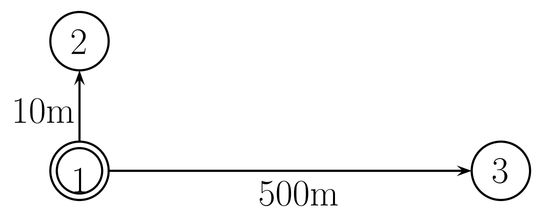


Figure 4.4: System demonstrating how some states can add more to obscurity than others

Many systems will have different properties or state relations that will determine how much a state will contribute to the obscurity of a marked state. The above example demonstrated this using distance, but these relations could be a large variety of things such as cost or traversability. By using an arbitrary function, the function can be designed to suit the properties of the given system or a certain system can be analyzed in many different ways through the application of different functions.

The functions that can be chosen as opacity weighting-functions must adhere to two constraints. The first is that w must map to non-negative real numbers. The second constraint is that it must be monotonically increasing with respect to subset inclusion. These constraints are there to reflect properties of opacity in systems. The first constraint, mapping to non-negative numbers, reflects how confusing more states together will never decrease the confusion of an observer. While it may not contribute to the obscurity, making any given state indistinguishable with a marked state will not reveal more information about the marked state. The second constraint is similar, it depicts how a set of indistinguishable states cannot be less obscure than any subset of the set, again reflecting how confusing more states is not detrimental. These constraints allow for useful properties to be leveraged when the function is used. An example of this is shown in Section 4.3 when an algorithm enforcing degree of opacity using supervisory control is presented.

While these constraints may seem restricting, since they were designed to reflect desirable properties of opacity problems, almost all functions one might wish to use as an opacity weighting-function fall within these constraints. What follows is a list of a variety of different functions that are valid opacity weighting-functions.

- Indicator Function: determines whether or not there is a secret state in the super-state A.

$$w(A) = \begin{cases} 1 & \text{If some condition is met} \\ & \text{(e.g., a non-secret state in } A\text{)} \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

  This function can be used to reflect the traditional notion of opacity.

- Counting Function: counts the number of states in the super-state A.

$$w(A) = |A| \tag{4.2}$$

- Distance Function: finds the state in A which will have the minimum distance to all other states in A then returns the distance to all states from that state.

$$w(A) = \min_{q \in A} \sum_{q' \in A} (d(q, q')) \tag{4.3}$$

  Other distance functions may be more useful than the one provided above. An example of such a distance function is applying the Bellman-Ford algorithm [15, 16] to determine the shortest path between the states in the super-state A.

- Edge-Cost or Traversability Function: given a function $f : Q \times Q \to \mathbb{R}_{\geq 0}$, which determines the weight of the edges comprising the shortest path between two states, it finds the state in A which will return the smallest sum of all path weights to all other states in A.

$$w(A) = \min_{q \in A} \sum_{q' \in A} (f(q, q')) \tag{4.4}$$

  The edge-cost function f could represent how costly it is to traverse between states. These costs could be incurred from difficult terrain, fuel costs, obstacles, and distance. For these functions to be valid opacity weighting functions, the edge-costs must be non-negative values.

In the rest of this thesis, if it is not explicitly stated, the opacity weighting-function that will be used is the counting function. It will be used most often because it is the simplest to understand and visually affirm, making it ideal for examples.

Now that the relevant security properties have been captured by the opacity weighting-function, this function can be applied to a state to capture the *degree* *of opacity of a state.*

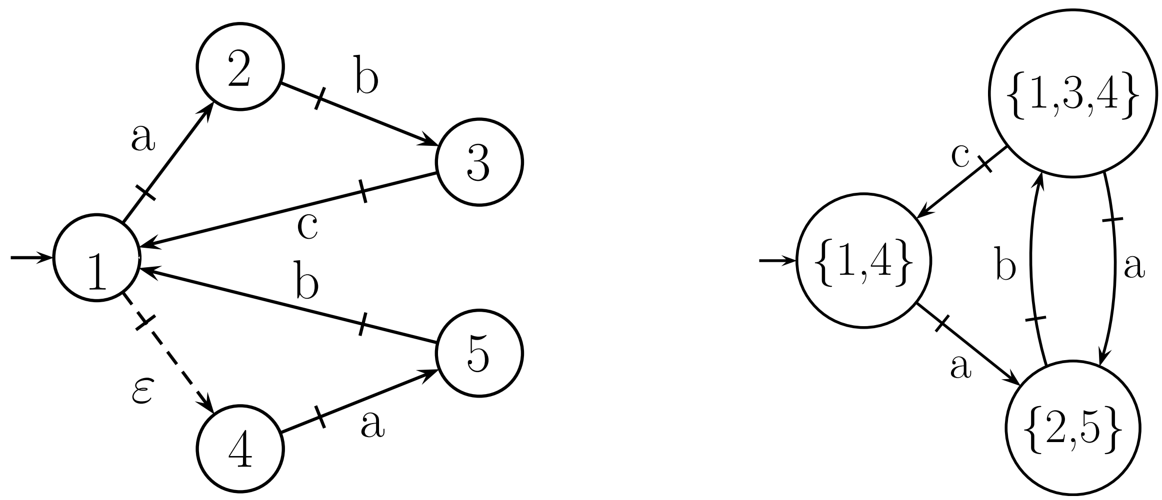
Definition (Degree of Opacity of a State)**.** *Suppose that we have a system* $G = (Q, \Sigma, q_0, \delta_G)$ *and an observable alphabet* $\Sigma_o \subseteq \Sigma$*. If we denote the observer automaton of* G *with respect to* Σ_o *as* $H = \mathrm{Obs}_{\Sigma_o}(G) = (\mathcal{A}, \Sigma, A_0, \delta_H)$*, then the degree of opacity of state* $q \in Q$ *is defined as follows:*

$$\Theta_{\Sigma_o}(q) = \min_{A \in \mathcal{A} \mid q \in A} w(A).$$

The degree of opacity of a state uses the opacity weighting function to capture how obscure or how opaque a given state is. The degree of opacity of a state, q, is determined by finding the adversarial estimate, A, that has the smallest level of obscurity out of all estimates that include state q. The level of obscurity of an estimate is calculated by applying the opacity weighting function. The degree of opacity of a state can be used in place of traditional notions of opacity to determine the level of obscurity of specific important states.

Consider the system in Figure 4.5a. Suppose we wish to calculate the degree of opacity of State 4 using the counting function as the opacity weighting function. First, calculate the observer to obtain the estimates of the system. The observer is presented in Figure 4.5b. State 4 is present in two different estimates: estimate $\{1, 3, 4\}$ and estimate $\{1, 4\}$. Using the counting function, the first estimate has $w(A) = 3$, and the second has $w(A) = 2$. By taking the minimum of these two weightings, we determine that the degree of opacity of State 4 is two.

As stated in Section 4.1, one weakness of traditional opacity is that it can only be used on systems with specific secret behaviour. The degree of opacity of a state has this same weakness. However, the degree of opacity of a state can be extended



(a) Example system (b) Observer for system in 4.5a

Figure 4.5: Example showing how to calculate the degree of opacity of a state

to the *overall degree of opacity* or the *degree of opacity of a system.*

Definition (Overall Degree of Opacity)**.** *Given a system* $G = (Q, \Sigma, q_0, \delta_G)$ *and an observable alphabet* $\Sigma_o \in \Sigma$*, the overall degree of opacity of the system with respect to the observable alphabet is defined as follows:*

$$\Theta_{\Sigma_o}(G) = \min_{q \in Q} \left[\Theta_{\Sigma_o}(q)\right].$$

The overall degree of opacity is equivalent to the minimum degree of opacity of all states in the system. Practically speaking, it is the minimum level to which all states are guaranteed to be obscured.

This can be shown practically by extending the example shown in Figure 4.5. To calculate the overall degree of opacity, the degree of opacity is calculated for every state in the system. The degree of opacity of all states are summarized in the Table 4.1.

State	Degree of Opacity
1	2
2	2
3	3
4	2
5	2

Table 4.1: Degree of opacity for each state in Figure 4.5

Since the minimum degree of opacity present of the states in Table 4.1 is two, the overall degree of opacity is two. Note that since we are using the counting function as the opacity weighting function, the calculation can be simplified to simply be the minimum size of the estimates.

Degree of opacity was designed specifically to target the weaknesses of traditional opacity. Where opacity is binary in nature, only opaque or transparent, and inflexible, degree of opacity uses a weighting function to quantify obscurity based on important system characteristics. Overall degree of opacity can also be applied to a whole system, allowing for the set of problems that lack specific secrets to be analyzed using opacity as well. While this expands the scope of opacity, it is worth noting that it is not necessary to calculate the degree of opacity over all states in the system. If a system does have a set of secrets, this framework can be applied by restricting the set of states that the degree of opacity is calculated over. This could also be a part of the design for the opacity weighting-function, like it is for the indicator function in Equation 4.1.

The following example demonstrates how degree of opacity can be used in situations where opacity cannot due to the weaknesses outlined above. It is a continuation of the maze search example shown in Section 4.1 in Figure 4.3. In this example, there are autonomous vehicles searching the maze modelled in Figure 4.3 for a target whose location is unknown. Both the location of the vehicles themselves as well as the location of the target need to be obscured to a certain degree while still allowing the agents to communicate some information to help the other agents. The states are connected with paths that take different times to traverse, these times are shown in hours in Figure 4.6. The adversary will find it too costly to search an estimate with a cumulative time cost of over one hour. In this case, the cumulative time cost of an estimate is the shortest time it takes to visit all states in the estimate. Does the communication map in Figure 4.7, allow for the target obscurity to be reached?

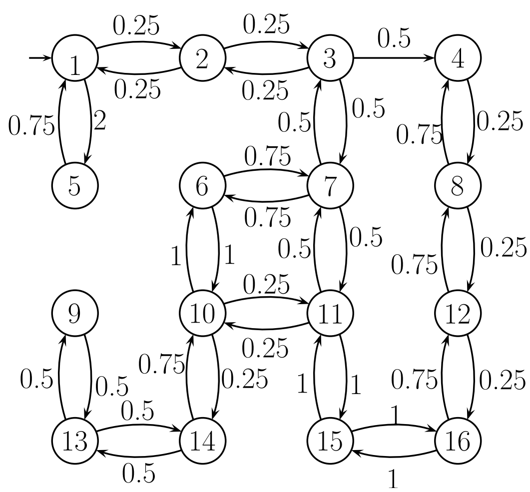


Figure 4.6: Depiction of traversability times for the system in Figure 4.3

The calculated observer of the system is shown in Figure 4.8. This observer shows the set of indistinguishable states as the upper part of the fractional notation with the cumulative search cost of the estimate in the denominator.

Through inspection, it is evident that the system does not reach the target overall degree of opacity since the estimate with State 4 and State 8 only has a cumulative time cost of 0.25 hours.

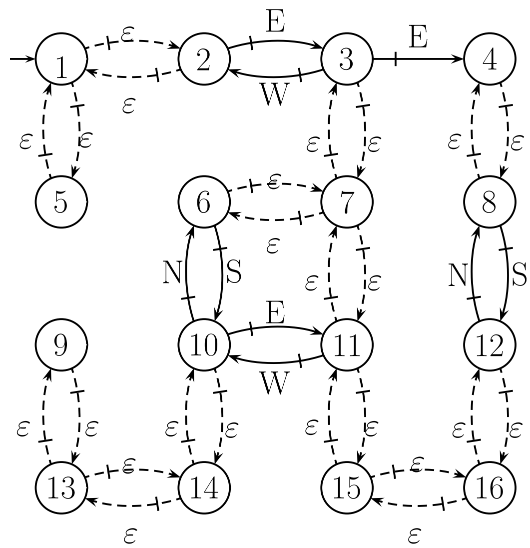


Figure 4.7: Communication map for the system in Figure 4.3

The above example shows that this version of opacity can be very useful. It could even be considered more useful than traditional notions because of its added applicability. In fact, it is a strictly more general definition of opacity. Consider CSO presented in Definition 2.2 of Chapter 2; with specific choices of an opacity weighting function, it can be expressed using the degree of opacity framework. In this case, the selected opacity weighting function is the indicator function (Equation 4.1) which will return a one when there is a non-secret state present. Using this opacity weighting function to calculate the degree of opacity, we perform the calculation over all secret states. The overall degree of opacity equation under these conditions is explicitly written out below where Q_S is the set of secret states.

$$\Theta_{\Sigma_o}(G) = \min_{q \in Q_S} \left[\min_{A \in \mathcal{A} \mid q \in A} \begin{cases} 0 & \text{if } A \subseteq Q_S \\ 1 & \text{otherwise} \end{cases} \right].$$

It was explicitly shown above that CSO is encapsulated by degree of opacity,

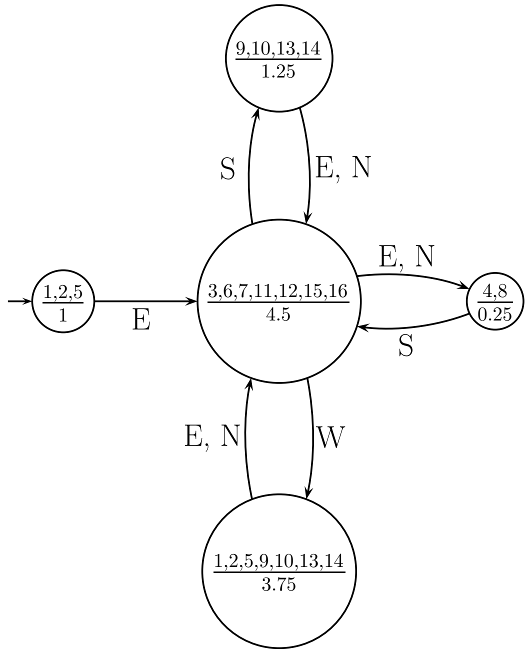


Figure 4.8: Observer for the system in Figure 4.7

but what about other definitions of opacity? It has been shown that many of the most commonly used versions of opacity are equivalent [7]. This specifically includes CSO, initial-state opacity, initial-and-final-state opacity, and language-based opacity. It cannot be guaranteed that degree of opacity can describe all types of opacity, however for definitions of opacity that are equivalent to CSO, the notion of "degree" that this framework introduces would be useful in adapting those versions to also be able to express levels of obscurity.

Now that the definition of degree of opacity has been established, the next logical step is to ask how it can be applied to security problems. Since degree of opacity is a more general definition of the most common definitions of opacity, it is possible that methods enforcing opacity can be extended to enforce degree of opacity. This is explored in the sections that follow.

4.3 Enforcing Degree of Opacity Using Supervisory Control

The material in this section is presented in the manuscript "Enforcing Degree of Opacity with Supervisory Control" [14]. This conference paper is in the process of being submitted for publication at a conference. This paper was written by myself and co-authored by Richard Hugh Moulton and Karen Rudie. The material in this section was joint work between Richard Hugh Moulton and myself. It is based on principles presented in an earlier work by Moulton et al. [17]. My specific contributions will be noted throughout the section.

Degree of opacity, like traditional opacity, is a measurement tool and as such is only useful in measuring the obscurity of a system. As it stands, there is no way to make use of degree of opacity in order to enhance the security of a system. To tie it back to the motivating problem, if we want to protect an autonomous vehicle's location as it is searching for a secret with an unknown location, we would want all positions that the vehicle can access to be obscured to a target level of obscurity. Degree of opacity can be applied to the system to say whether or not this level has been achieved, but if the level of obscurity is not reached, degree of opacity does not dictate what can be done to reach the desired obscurity. As such, it would be useful to develop a process that enforces a target degree of opacity by manipulating parameters of the system.

There are many possible ways to interact with discrete-event systems. If you wish to control the behaviour of the system, one popular method is to use supervisory control. The main principles and features of supervisory control were introduced in Chapter 2. This method of control enacts control decisions, either "enable" or "disable", on every controllable event in the system. By judiciously choosing which events to enable or disable, the system can be guided towards desirable outcomes or alternatively, away from undesirable situations. Supervisory control has been used to enforce traditional notions of opacity, like CSO. The first appearance of this occurred in Dubreil et al. [18]. In Moulton et al. [17], the notion of subobservers was presented and used to implement a more efficient method of supervisory control that enforces opacity. As shown in the previous section, degree of opacity captures traditional notions of opacity so it is possible that algorithms enforcing traditional opacity can be modified to enforce degree of opacity. This section explores this by showing the work from Schonewille et al. [14] which builds upon the method from Moulton et al. [17] to enforce degree of opacity.

We begin by explicitly defining the problem of enforcing degree of opacity. The problem definition is provided below.

Problem 3 (Degree of Opacity-Enforcement Problem)**.** *Given a system $G = (Q, \Sigma, q_0, \delta_G)$, an observable alphabet $\Sigma_o \subseteq \Sigma$, and a desired degree of opacity μ, synthesize a supervisor S such that*

$$\Theta_{\Sigma_o}(S/G) \geq \mu.$$

To ground this mathematical definition, apply this problem statement to the motivating situation in Problem 1. By framing this situation with the above problem statement, it states that we wish to synthesize supervisor S that will control where the vehicles can go so that they can only access states that will at least be obscured to the target degree of opacity.

As is evident, the solution to this problem is the supervisor that we hope to synthesize. This will be done by modifying the algorithm that produces a supervisor enforcing opacity in Moulton et al. [17]. This algorithm requires a few structures and definitions before it can be implemented.

The first structure that is needed is the augmented I-observer. It is the parallel composition of a system and its observer (as presented in Chapter 2), however states are marked when they violate the relevant opacity condition [19]. The structure is denoted by

$$M = G \mid\mid H = (X, \Sigma, x_0, \delta_M, X_m).$$

For a desired degree of opacity, μ, X_m is defined to be

$$X_m := \{(q, A) \mid w(A) < \mu\}.$$

The resultant system is composed of states that show where the system currently is as well as the estimate of where the system is from the perspective of an external observer. This structure is very useful in visualizing the evolution of the observation of the system since it retains the relations of the base system.

The other structure needed is the *subobserver* which was presented in Moulton et al. [17].

Definition (Subobserver)**.** *We say that the automaton* $H' = (\mathcal{A}', \Sigma, A'_0, \delta_{H'})$ *is a subobserver of* $H = (\mathcal{A}, \Sigma, A_0, \delta_H)$*, denoted* $H' \mathrel{\tilde{\sqsubseteq}} H$*, if*

$$\delta_{H'}(A'_0, s) \subseteq \delta_H(A_0, s)\ \forall s \in L(H').$$

An observer automaton is a subobserver of a given observer if the behaviour of the observer automaton is a subset of the behaviour of the given observer. This subobserver property is an extension of the subautomaton property in Cassandras and

Lafortune [2, p. 86]. The subobserver property is very important in decreasing the computational complexity of the opacity enforcing algorithm. Through the use of this property, Moulton et al. [17] developed the `REFINE` algorithm which refines an augmented I-observer to produce an updated augmented I-observer $M' = G' \mid\mid H'$. Using this algorithm that is based on the subobserver property allows for the augmented I-observer to be updated without the need for the updated observer, H', to be computed. The computational complexity of calculating an observer is exponential with respect to the number of states, since it is based on NFA-to-DFA conversion [20]. This is very computationally intensive however its replacement, the `REFINE` algorithm, was shown to be linear with respect to the number of states in $G \mid\mid H$ [17].

The algorithm that was presented in Moulton et al. [17] which enforces degree of opacity using the subobserver property can be found in Appendix A. This algorithm was adapted in Schonewille et al. [14] and is presented as Algorithm 1.

Data: System $G = (Q, \Sigma, q_0, \delta_G)$, the adversary's observable events, $\Sigma_o \subseteq \Sigma$, and a degree of opacity, μ.
Result: A supervisor S such that $\Theta_{\Sigma_o}(S/G) \geq \mu$, or notification that no solution exists.

Compute $H = \mathrm{Obs}_{\Sigma_o}(G)$;
Compute $M = G \mid\mid H$;
Mark all states $x = \frac{q}{A}$ in M where $w(A) < \mu$;
while *M has marked states* **do**
    if $\max_q \Theta_{\Sigma_o}(q) < \mu$ **then**
        return No solution exists ;
    Set Δ equal to the marked states in M ;
    $M =$`REFINE`(M, Δ);
    Mark all states $x = \frac{q}{A}$ in M where $w(A) < \mu$;
    if *All states in M are marked* **then**
        return No solution exists ;
return $S = M$;

Algorithm 1: Supervisor synthesis for Problem 3

This algorithm marks any state that does not have a degree of opacity equal to or above the target degree of opacity, μ. Subsequently, the algorithm disables all transitions leading uncontrollably to each marked state. The `REFINE` algorithm from Moulton et al. is then used to update the augmented I-observer. Finally, if any states still do not reach the target degree of opacity, they are marked and the algorithm performs another iteration. If there are no more marked states or no more transitions, then the algorithm terminates with a valid degree of opacity enforcing supervisor or, respectively, the answer that the system cannot enforce the target degree of opacity. It is worth noting that this algorithm enforces an overall degree of opacity, however it is easily adapted to only obscure a set of secret states to a target degree. This can be done by only checking the degree of opacity of states that include a secret state in their estimate.

For this algorithm to be useful, it must be proved to be correct. To accomplish this, Schonewille et al. [14] presented a necessary lemma. This lemma shows that when an event to a state is disabled, the size of the adversary's estimate does not increase. This implies that the opacity weighting of that estimate also does not increase.

Lemma 1 (The effect of disabling a transition on the Degree of Opacity of an arbitrary state)**.** *Suppose we have an augmented I-observer $M = (X, \Sigma, x_0, \delta_M)$, a state $x = \frac{q}{A} \in X$, and a state $x_m = \frac{q_m}{A_m} \in X$, generate $M' =$REFINE$(M, \{x_m\})$. If the corresponding state of x in M' is still accessible, then we denote it $x' = \frac{q}{A'}$ and we have that $w(A') \leq w(A)$.*

Proof. We denote the denominator portions of M and M' (i.e., the observer automata) as H and H' respectively.

From the subobserver property (Definition 4.3, [17]), since $H' \tilde{\sqsubseteq} H$ we know that the state labels in every estimate in H' are a subset of the state labels in the corresponding estimate of H. This means that $A' \subseteq A$. Since the opacity weight function is monotonically increasing with respect to the containment relationship, this directly implies that $w(A') \leq w(A)$. □

Initially the result of Lemma 1 seems to counteract the result that the algorithm aims to produce. Fortunately, this is not the case. As will be shown below in the proof of Theorem 2, any events that are disabled are necessary in order to synthesize a degree of opacity enforcing supervisor. Events are only disabled when they lead uncontrollably to a marked state. If this disabled event causes the degree of opacity of another state to decrease, it is inconsequential since the original marked state needed to be made inaccessible. Since we are only employing supervisory control, this is the only method that will prevent the system from reaching that marked state while restricting the minimum amount of system behaviour. The other important property that Lemma 1 provides is monotonicity. Since disabling an event can never increase the opacity weighting of a state, it is impossible to increase the opacity weighting of an event while only using supervisory control. This means that marked states must be made inaccessible, and if done in a minimal fashion, no other set of disabled events will produce a better result.

With Lemma 1, the proof of correctness for Algorithm 1 can now be presented. This proof aims to show that Algorithm 1 does indeed enforce the target degree of opacity if possible, and that the synthesized supervisor is minimally restrictive to the behaviour of the system.

Theorem 2 (Correctness of Algorithm 1)**.** *Consider a system* $G = (Q, \Sigma, q_0, \delta_G)$*,* *an observable alphabet* $\Sigma_o \subseteq \Sigma$*, an observer* $H = \text{Obs}_{\Sigma_o} = (\mathcal{A}, \Sigma, A_o, \delta_H)$*, and a minimum desired degree of opacity,* μ*. Algorithm 1 produces a minimally restrictive supervisor* S *such that* $\Theta_{\Sigma_o}(S/G) \geq \mu$ *or returns that no such supervisor exists.*

Proof. Algorithm 1 can terminate for three reasons: the stopping condition at Line 6 is met; the stopping condition at Line 11 is met; or an automaton is returned as the minimally restrictive supervisor. We will consider these cases separately.

If the algorithm terminates at Line 6, then the maximum degree of opacity for a state in G is less than μ. Since we cannot increase the degree of opacity of an individual state through supervisory control (Lemma 1), this maximum degree of opacity is an upper-bound on the overall degree of opacity.

If the algorithm terminates at Line 11, then allowing the system to reach any remaining state in the augmented I-observer will cause the degree of opacity constraint to be violated. All remaining states must therefore be made inaccessible, which means that the null automaton will be returned.

If Algorithm 1 returns a supervisor S, it remains to be shown that it enforces the desired overall degree of opacity and is minimally restrictive.

The supervisor S enforces the overall degree of opacity.

We proceed by contradiction. We assume that Algorithm 1 returned S as a supervisor, but that S does not enforce the overall degree of opacity. This means there exists a state $q \in Q$ such that $\Theta_{\Sigma_o}(q) < \mu$. This means that there exists an adversary estimate $A \in \mathcal{A}$ such that $q \in A$, $w(A) < \mu$, and $x = \frac{q}{A} \in X$. If this is the case, however, state x would have been marked at Line 3 or 9 and would have been made inaccessible in the next iteration of the while loop. This particular automaton S would therefore not be returned by Algorithm 1, which leads to a contradiction.

The supervisor S is minimally restrictive.

We proceed by contradiction. Assume that a supervisor S' exists that enforces the overall degree of opacity and is less restrictive than S. This means that there is (at least) one transition in G that S disables that S' does not. However, transitions are only disabled by S if they have been added to the list Δ during an iteration of the while-loop in Algorithm 1. These transitions either lead directly to a marked state $x = \frac{q}{A}$, which means that $w(A) < \mu$, or lead uncontrollably to such a marked state. If S' enables any of these transitions, it will permit a particular string to occur in the system that leads to an adversary estimate whose degree of opacity is less than μ and therefore S' does not enforce the overall degree of opacity, which is a contradiction.

We therefore conclude that Algorithm 1 produces a minimally restrictive supervisor that solves Problem 3 or returns that no such supervisor exists. $\square$

Now that Algorithm 1 has been proven to be correct, I will demonstrate how the algorithm is put into practice. I developed this example for the paper by Schonewille et al. [14], although it does not appear in the final version, and it is inspired by the traditional DES problem, the cat and mouse problem [21]. For this cat and mouse problem there is a mouse who is searching a series of rooms and corridors for food. The mouse is trying to do so without being caught by the cat who is lurking nearby. Certain corridors are rigged with different alarm bells which would alert the cat where the mouse could be. The room system is shown in Figure 4.9. One thing to note is that the corridor between Room 9 and Room 10 is slanted so the mouse will move uncontrollably from Room 9 to Room 10.

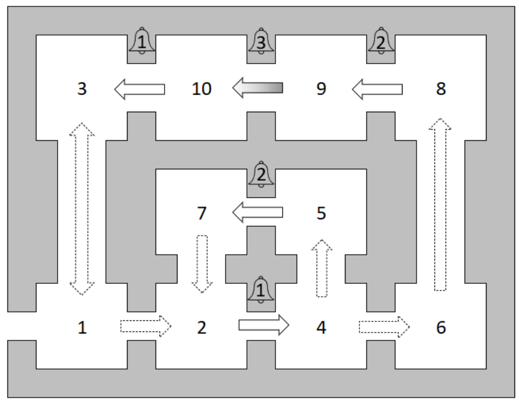
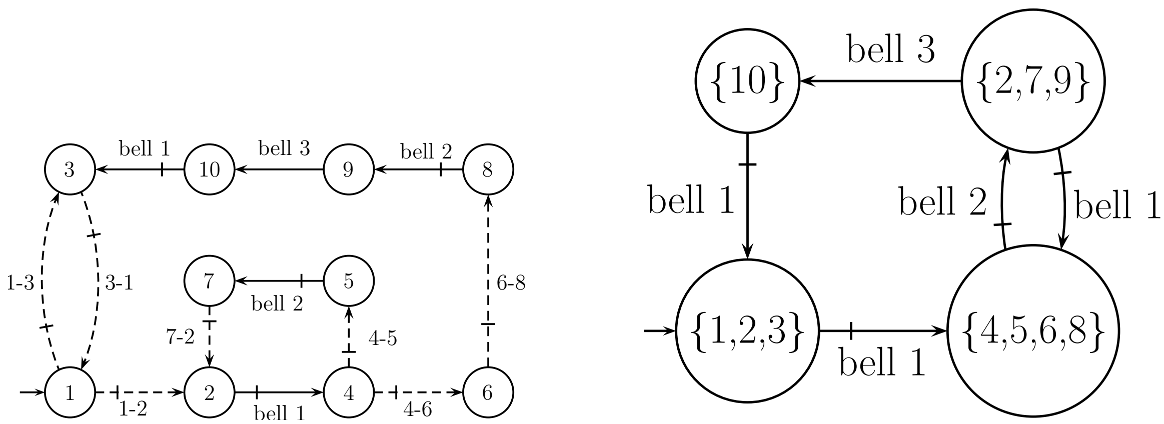


Figure 4.9: Room and hall system for the cat and mouse problem

One additional piece of information is that the cat is lazy and will only go to search for the mouse if it only has to search two or fewer locations. Since the mouse's location is the "secret" to be protected, there is no set of secret states so CSO cannot be used. With this property, traditional opacity cannot be enforced. Let this be formulated instead as a degree of opacity enforcement problem. As an initial set-up step, the room system is translated to an automaton. The equivalent system and its accompanying observer are displayed in Figure 4.10a and Figure 4.10b respectively.

Since there is no set of secret states, we wish to enforce on overall degree of opacity. With degree of opacity, the first consideration is the choice of opacity weighting-function. The opacity weighting-function that will be used is the counting function, $w(A) = |A|$. Since this is a small group of rooms and to properly reflect the cat's behaviour, this is the most appropriate weighting-function for the situation. If the system was larger, it would likely be more beneficial to use a distance function instead. For this example, the target degree of opacity to be enforced is $\mu = 3$ since the cat



(a) Automaton modelling the room system shown in Figure 4.9

(b) Observer for system in 4.10a

Figure 4.10: Automata modelling the cat and mouse problem

will only search if it can distinguish the mouse's location between two or fewer states. With these parameters, employ Algorithm 1 to choose where the mouse can search for food while also avoiding the cat's detection. To initialize the algorithm, the system's augmented I-observer must be constructed. This automaton can be seen in Figure 4.11.

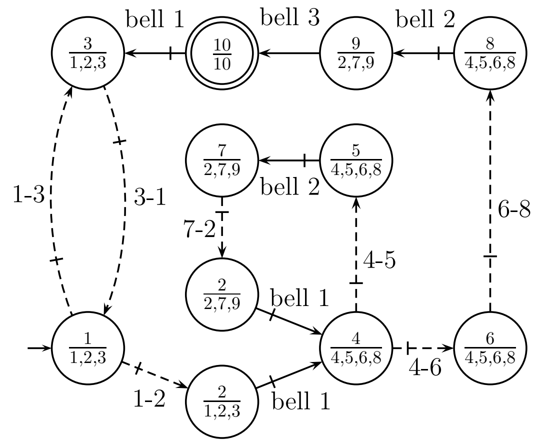


Figure 4.11: Initial augmented I-observer for the cat and mouse problem

Through a quick inspection, it can be seen that the overall degree of opacity is one. This is because the state $\frac{q}{A} = \frac{10}{\{10\}}$ has a weighting of one. This is currently the only state that does not reach the target degree of opacity of three. In the first iteration of the algorithm, this will be the only marked state. It is worth noting that the state representing Room 2 has split into two different states in the augmented I-observer. This is a phenomenon called “state-splitting” [17]. State splitting occurs since this particular room has different estimates depending on how it is reached. The algorithm must now choose which event to disable to restrict access to this state. Since the corridor leading from Room 9 to Room 10 is uncontrollable, the corridor prior must be disabled. This means that the hall between Room 8 and Room 9 will be made inaccessible. After this decision is made, the augmented I-observer can be refined. The updated augmented I-observer is presented in Figure 4.12.

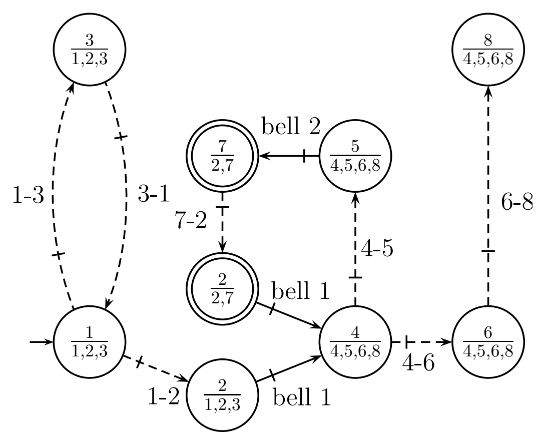


Figure 4.12: Augmented I-observer for the cat and mouse problem after the first algorithm iteration

The algorithm then proceeds to mark any states that do not reach the target degree of opacity. Two other states in the augmented I-observer relied on the presence of Room 9 to reach a degree of opacity of 3. Now that Room 9 has been made inaccessible, these states are no longer obscured to the necessary degree and are marked. Since there are still marked states, the algorithm performs a second iteration. In this iteration, the algorithm chooses to disable the hall between Room 7 and Room 2 as well as the hall between Room 5 and Room 7. The augmented I-observer is then recalculated, it is presented in Figure 4.13.

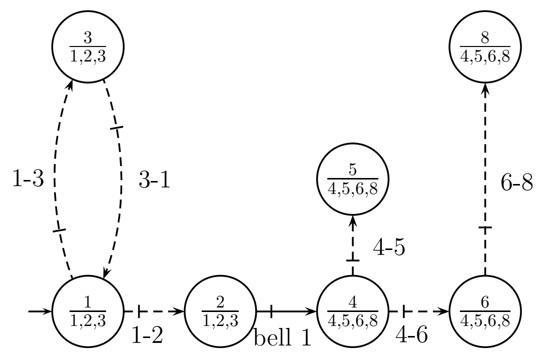


Figure 4.13: Augmented I-observer for the cat and mouse problem after the second algorithm iteration

By making these two halls inaccessible, all states now reach the target degree of opacity. No new states are marked so the algorithm terminates. Since the algorithm has terminated, the augmented I-observer in Figure 4.13 describes the greatest amount of halls and rooms that the mouse can traverse while still maintaining an overall degree of opacity of three.

In addition to the cat and mouse example, we will demonstrate the algorithm on a small scenario based on the motivating industry problem 1. This example demonstrates how a different weighting function can be utilized to calculate a scenario-specific notion of security. Suppose a group of autonomous vehicles are sent into an area of known terrain to find a target whose exact location is unknown. The vehicles' are attempting to locate the target as fast as possible. To facilitate this, they will communicate to the other vehicles when they cross from one region of the terrain to another. Unfortunately, an adversary is monitoring these global communications and aims to determine the location of the vehicles and the target.

Figure 4.14 depicts the terrain; it is a series of clearings connected by different paths. The clearings are numbered and divided into three regions. Region one, $R1$, contains clearings $1, 2, 3, 4$. The second region, $R2$, consists of clearings $5, 6, 9, 10$, and the final region, $R3$, is the pair of clearings $7, 8$. Since the adversary is listening to the vehicles' communications, they will be aware of movement between these sets of clearings. However, this is a communication dead-zone in clearing 3 so the change of regions between clearings 3 and 7 as well as clearings 3 and 5 are invisible. Taking this into consideration, the adversary will specifically be aware of the vehicles' movements between clearings: 1 and 8, 4 and 9, 6 and 7, as well as 10 and 7.

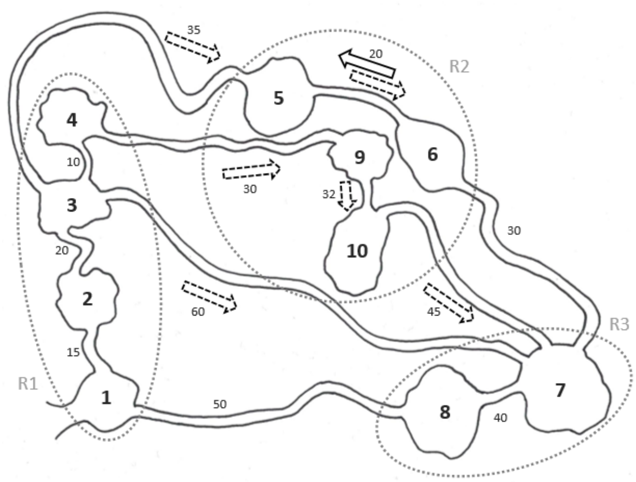


Figure 4.14: Diagram of the terrain with traversability times, terrain constraints, and regions

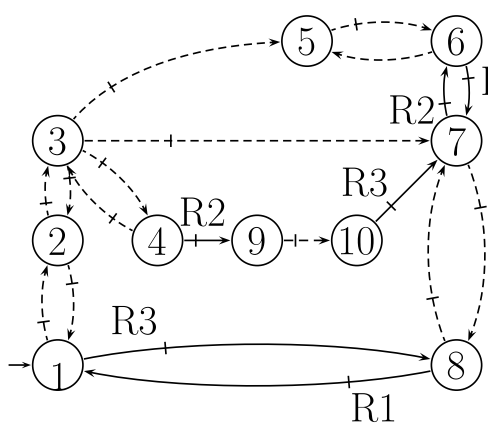
Each path is labelled with a traversability time (Figure 4.14). This time is determined from the path's length and terrain features. In addition, some paths have major terrain features which change how the vehicles can interact with those paths. In Figure 4.14, paths marked with an arrow cut through steep areas so the vehicles can only travel the path in the marked direction. Also the path with the solid arrow from clearing 6 has water running through it, pushing the vehicles uncontrollably to clearing 5.

Since the secrets we wish to protect are the vehicles' positions and the position of the target, we cannot apply CSO since the target and vehicles could be in any given state at a time, they are not a fixed set of secrets. We shall instead enforce an overall degree of opacity on the entire system.

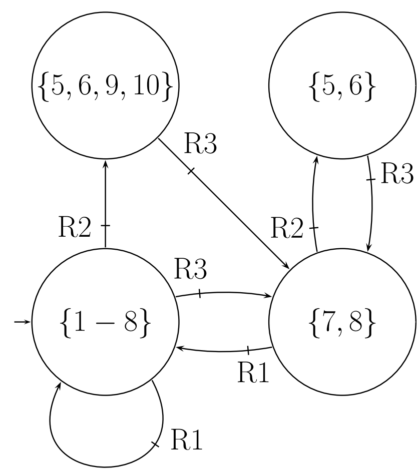
Suppose that the adversary has a small autonomous vehicle of its own. The adversary will deploy this agent if the vehicle will not need to spend more than 30 minutes pinpointing the location of an agent or the target. We are not certain where the adversary will deploy its agent so we will consider the search time to be the shortest path between the clearings that they estimate the target or vehicle will be. In the worst case scenario, the adversary will be able to choose its starting point and will choose the clearing which will have the shortest path between the clearings they believe the vehicle or the target is. We can use these considerations to design the opacity weighting function and target degree of opacity for our situation. The opacity weighting function will be the shortest time to traverse a given group of estimated location. In other words, the search time of an estimate is calculated by determining the shortest path that visits all states in an estimate and setting the degree of opacity as the time to traverse this path. If the target was estimated to be in one of clearings 3, 4, 5, and 7, the search time, and hence the degree of opacity of the estimate, is 95 minutes. This is the time to traverse from clearing 4, to clearing 3, to clearing 5, then clearing 6, and finally clearing 7, the shortest path traversing clearings 3, 4, 5, and 7. Since the adversary only deploys its vehicle if the search time is less than or equal to

30 minutes, we can set our target overall degree of opacity to $\mu = 35$. The buffer of 5 minutes will ensure that the adversary will not choose to deploy their vehicle.

In order to apply Algorithm 1 to synthesize a degree of opacity enforcing supervisor, the system must be transformed into an automaton. This automaton, G, is shown in Figure 4.15a. The first step of the algorithm is to take this system and generate the corresponding observer, H, shown in Figure 4.15b. These two automata are then combined to for the system's augmented I-observer, M, shown in Figure 4.16.



(a) System G



(b) Observer H

Figure 4.15: Automaton and observer for the autonomous vehicle search problem

The degree of opacity is calculated for all of the estimates in the augmented I-observer. Only one estimate does not achieve the desired target degree of opacity of $\mu = 35$. The estimate with only clearings 5 and 6 has search time of 20 minutes and is the system's overall degree of opacity. The algorithm marks the states responsible for this lowered degree of opacity, $\frac{5}{5,6}$ and $\frac{6}{5,6}$, denoted by the double rings. It is worth noting that there are multiple states representing each of the clearings 5, 6, 7, and 8. This is a phenomenon called state splitting and it arises from different observable

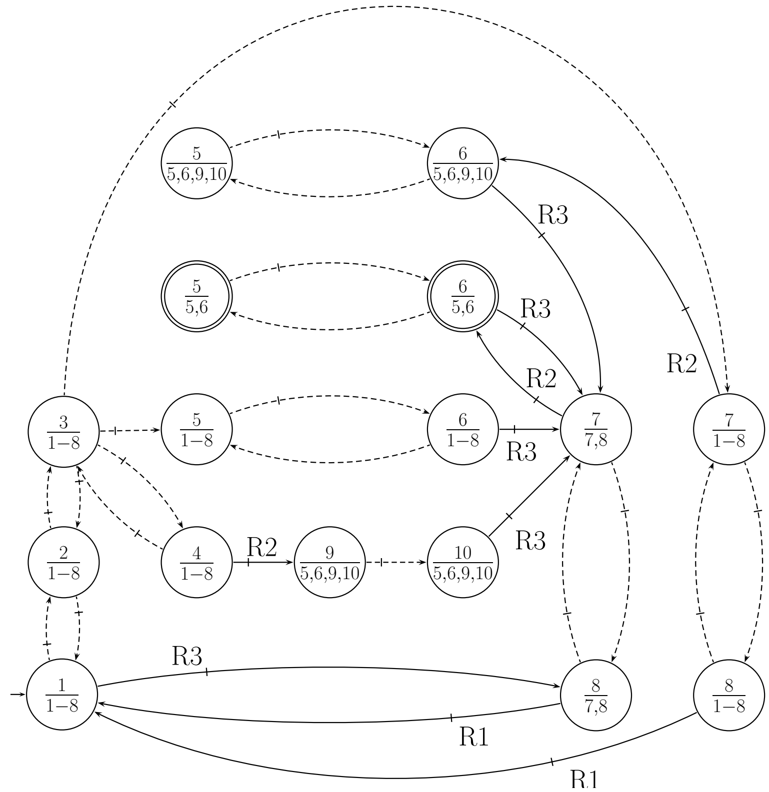


Figure 4.16: Augmented I-observer M of system G in Fig. 4.15a and observer H in Fig. 4.15b

paths existing that result in different estimates. The first iteration of Algorithm 1 produces the refined augmented I-observer in Fig. 4.17 by making the states $\frac{5}{5,6}$ and $\frac{6}{5,6}$ inaccessible. Since the path from clearing 6 to clearing 5 is uncontrollable, the path leading to it must be disabled instead. Fortunately, this is the path leading from clearing 7 to clearing 6, the path that we already need to disable.

Unfortunately, the prior control action has a detrimental effect to the obscurity of clearings 9 and 10 since clearings 5 and 6 are no longer accessible and hence no longer indistinguishable with clearings 9 and 10. The degree of opacity for this estimate is recalculated and found to be only 32 minutes. Since this does not reach the target degree of opacity, it is marked and the algorithm makes the states $\frac{9}{9,10}$ and

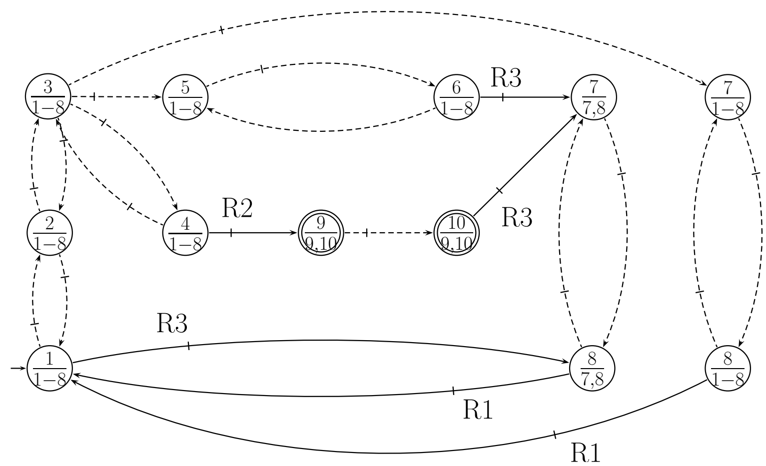


Figure 4.17: Augmented I-observer M after the first iteration

$\frac{10}{9,10}$ inaccessible.

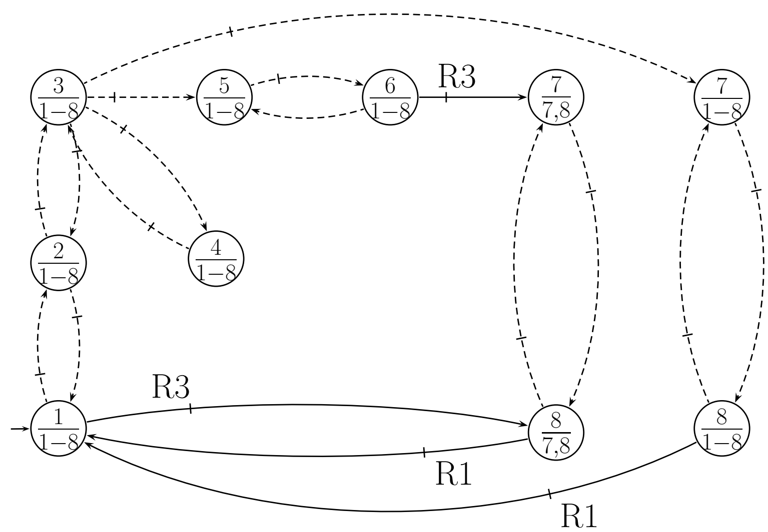


Figure 4.18: Augmented I-observer M after the second iteration

After this control action is taken, the updated augmented I-observer, Figure 4.18, contains no marked states so it is a valid minimally restrictive supervisor for our system that enforces an overall degree of opacity of $\mu = 35$. While this supervisor is minimally restrictive, in this situation it does not permit access to all clearings. This could become an issue if the target we are searching for is, in fact, in clearing 9 or 10. This is a problem inherent to supervisory control in general: by making some strings inaccessible, there is a chance that we may make desirable system behaviour impossible to achieve. In this example, we could choose to relax our target degree of opacity to $\mu = 31$. While this leaves less of a buffer between our planned level of obscurity and when the adversary deploys its vehicle, it will allow for a system with all clearings to be accessible. With these new parameters, after the first iteration of the algorithm, states $\frac{9}{9,10}$ and $\frac{10}{9,10}$ in Figure 4.17 will not be marked since the search time is 32 minutes. This will halt the algorithm, causing Figure 4.17 to be the valid supervisor enforcing $\mu = 31$. All clearings in Figure 4.17 are accessible but certain path must be avoided, as dictated by the supervisor, to maintain a degree of opacity of 31.

While the algorithm works well for the above situations and is easy to apply, utilizing supervisory control can be restrictive, as demonstrated in the above example. With an unknown target location, it is very possible that the target will be in one of the states that are being made inaccessible. While the algorithm will still protect the agents' locations, it may now be impossible for them to find the target. For applications like this, a different technique to enforce degree of opacity may be more useful.

4.4 Enforcing Degree of Opacity Using Observability

As discussed in the previous section, supervisory control can be overly restrictive on a system, to the point where the system goals cannot be met. This necessitates a different strategy to enforce degree of opacity. The next method that will be presented was directly inspired by the motivating problem. The agents in this system are communicating to provide information about the search to other agents. What if this communication is leveraged instead? This technique changes what is observable in the system instead of what is accessible, allowing for all states to still be reached.

Properties of a system that are observable are the only information that external observers are allowed to access about the system. Alongside controllability, it is one of the major facets of discrete-event systems. The classic notions of observability presented in Chapter 2 are based on events. Events are transition labels and many different transitions can be instances of a given event. In general, all instances of an event are taken to have the same properties. If we consider controllability for example, if there is an event $\alpha \in \Sigma$, then $\alpha \in \Sigma_c$ or $\alpha \in \Sigma \backslash \Sigma_c$. This means that one instance of α cannot be uncontrollable if $\alpha \in \Sigma_c$. This may not pose an issue for many systems since this property may be inherent to the system or events can be broken into multiple versions and relabelled for each variation of the properties. However, for systems where the observation of a transition is the event label, you cannot split the events since the labels since potential indistinguishability of transitions may be lost. In this case, it is more useful to define properties based off of the system's transitions instead.

The relevant definitions for a transition-based framework are presented below. While these concepts are familiar to the DES field, appearing in various sources [22, 23], I have adapted the symbology and definitions to suit this specific application.

Definition (Set of Transitions)**.** *The set of transitions, T, is the set of all valid transitions in a system G.*

$$T = \{t \mid t =< q_1, \sigma, q_2 >,\ \delta(q_1, \sigma)! \ \wedge\ \delta(q_1, \sigma) = q_2\}$$

To mimic strings of events, transitions can be put together in sequences. Each valid sequence is called a *trace*.

Definition (Set of Traces)**.** *The set of traces, $trace(G)$, is the set of all valid sequences of transitions in system G, written in a compact form. Sequences of transitions are written as $q_0\sigma_0q_1\sigma_1q_2...q_{n-1}\sigma_{n-1}q_n$ where $< q_i, \sigma_i, q_{i+1} >= t_i$ for any $i \in \{0, 1, ..., n-1\}$ and $t_0t_1...t_{n-1}$ is a sequence of transitions in the system. The trace format obviates the need for states to be written twice, back to back. It removes the duplicated states since it can easily be inferred that the resultant state of a transition becomes the beginning state for the next transition in the sequence. Since trace notation can easily be extended to a sequence of transitions, an element $s \in trace(G)$ can be referred to as $s = q_0\sigma_0q_1\sigma_1...q_n$ (trace notation) or as $s = t_0t_1...t_{n-1}$ (transition notation) so both transitions as well as state and events can be accessed from a trace. As a slight abuse of notation, when $t \in s$ is written, where $t \in T$ and $s \in trace(G)$, let this refer to when the transition t is a transition present in the sequence of transitions that trace s refers to.*

All of these traces start at state q_0 which is the initial state of the system G. Also, for a trace $s \in trace(G)$ let the final state of trace s be $q_f(s)$ (i.e., for $s = q_0\sigma_0q_1\sigma_1...q_n$, $q_f(s) = q_n$).

$$\begin{aligned} trace(G) = \{q_0\ \sigma_0\ q_1\ \sigma_1...q_n | (q_i\ \in Q\ \forall i = 0, 1, ..., n)(\sigma_j\ \in \Sigma\ \forall j = 0, 1, ...n-1) \\ (t_j =< q_j, \sigma_j, q_{j+1} >\in T\ \forall\ j = 0, 1, ...n-1)\} \end{aligned}$$

Just like the set of observable events, Σ_o, in this set-up there is a set of observable transitions.

Definition (Set of Observable Transitions). *The set of observable transitions is denoted by T_o where $T_o \subseteq T$. These are all of the transitions that are can be observed to occur when they are executed in the system.*

The set of observable transitions can be used to replace the use of Σ_o in various definitions. For example, when constructing an observer in this framework, it is denoted by $H = \mathrm{Obs}_{T_o}(G)$ and built by retaining only the observable transitions. The notation for degree of opacity is changed in the same way. Since the calculations will instead be based upon T_o, the degree of opacity of a state is $\Theta_{T_o}(q) = \min_{A\in\mathcal{A} \mid q\in A} w(A)$ and the overall degree of opacity is $\Theta_{T_o}(G) = \min_{q\in Q}\left[\Theta_{T_o}(q)\right]$. A projection can also be defined to project out unobservable transitions from a trace.

Definition (Observable Transition Projection). *The observable transition projection takes a sequence of transitions and outputs the corresponding sequence consisting only of observable transitions.*

$$P_{T_o} = T^* \to T_o^*$$

$$P_{T_o}(\epsilon) := \epsilon$$

$$P_{T_o}(t) := \begin{cases} t & t \in T_o \\ \epsilon & t \in T\backslash T_o \end{cases}$$

$$P_{T_o}(st) := P_{T_o}(s)P_{T_o}(t)$$

This projection can be used in conjunction with the event projection, Definition 4.4, to project out the observation of a trace.

Definition (Event Projection). *The event projection, P_Σ, is a function which takes a sequence of transitions and returns the event string.*

$$
\begin{aligned}
P_\Sigma &= T^* \rightarrow \Sigma^* \\
P_\Sigma(\epsilon) &:= \epsilon \\
P_\Sigma(t) &= P_\Sigma(q\sigma q') := \sigma \\
P_\Sigma(st) &:= P_\Sigma(s)P_\Sigma(t)
\end{aligned}
$$

Each observation can be associated with multiple traces. This is captured by the structure presented in Definition 4.4.

Definition (Set of Associated Traces of an Observation). *The set of associated traces of an observation, $tr(p_{obs}, T_o)$, is the set of traces which share the observation $p_{obs} \in \Sigma^*$ given the set of observable transitions T_o.*

$$
\begin{aligned}
tr &: \Sigma^* \rightarrow trace(G) \\
tr(p_{obs}, T_o) &= \{p \ \in trace(G) \ | P_\Sigma(P_{T_o}(p)) = p_{obs}\}
\end{aligned}
$$

Each trace leads to a state in the plant. An observation can be associated to a set of states by mapping all traces associated with that observation to their resultant states. This set of states is the state estimate of an external observer when that observation is perceived.

Definition (Set of Associated States of an Observation)**.** *The set of associated states of an observation, $Q(p_{obs}, T_o)$, is the set of states which can be reached by traces which have the observation p_{obs} given a set of observable transitions T_o.*

$$Q(p_{obs}, T_o) = \{q \ \in Q \mid \exists \ p \in tr(p_{obs}, T_o) \ \textit{such that} \ q_f(p) = q\}$$

The initial algorithm I developed to enforce an overall degree of opacity by concealing events is seen in Algorithm 2. This pseudocode represents a high-level view of the overall algorithm.

Data: System $G = (Q, \Sigma, q_0, \delta_G)$, the adversary's observable transitions, $T_o \subseteq T$, an opacity weighting-function, w, and a degree of opacity, μ.

Result: A set of transitions to conceal in order to achieve the target degree of opacity or the notification that concealing all transitions will not achieve the target

```
Obs_{T_o}(G) = (𝒜, Σ, A_0, δ_H);
G = Obs_{T_o}(G) Calculate current overall degree of opacity
  Θ_{T_o}(G) = min_{q∈Q} [Θ_{T_o}(q)] .;
Define a set of transitions to be concealed C = ∅;
while Θ_{T_o}(G) < μ and |𝒜| > 1 do
    for a ∈ 𝒜 do
        if w(a) = Θ_{T_o}(G) then
            a_min := a ;
            break ;
    for a ∈ 𝒜 such that there exists σ ∈ Σ where δ_H(a, σ) = a_min or
      δ_H(a_min, σ) = a do
        a'_min = ∞ if w(a) < a'_min then
            a'_min = a
    Find the transitions connecting a_min and a'_min in G.;
    Add the found transitions to the set of transitions to conceal C;
    Recalculate the observer Obs_{T_o}(G) where T_o = T_o ∪ C ;
    Recalculate the overall degree of opacity Θ_{T_o}(G) = min_{q∈Q} [Θ_{T_o}(q)] .
      where T_o = T_o ∪ C ;
return C;
```

Algorithm 2: Initial Algorithm Enforcing DoO through Concealment

This algorithm compiles a set, C, of transitions that need to be concealed in order to reach the target degree of opacity. It chooses these transitions by iteratively finding the estimate with the smallest weight, then removing the transition that connects this state to the smallest state of the ones it is connected to. The method drew inspiration from the construction of overall degree of opacity formula. Since the calculation of overall degree of opacity is based off of a minimum function, it is natural to target the minimum-weighted estimate since it would be the estimate that is holding back the degree of opacity the most. Concealing the transition that connects this estimate to the smallest of its connections was done in an attempt to promote minimality of this algorithm. By removing this specific transition, it would hopefully combine the states, boosting the weight of the minimum-weighted estimate as well as that of the smallest connected state. If the connected state would have violated the target degree of opacity condition, this would hopefully also boost this estimate to reach the target degree of opacity.

When discussing the efficacy of this algorithm, a minimality constraint should be a part of this discussion. This type of constraint will allow for the least amount of information to be concealed, meaning friendly observers will have as much information as can be afforded. Since this situation could produce comparable solutions, there is no definite minimal solution. In this case, the algorithm would be considered minimal if no subset of the set of transitions being concealed could cause the system to reach the target degree of opacity. Unfortunately, while Algorithm 2 does indeed enforce the target degree of opacity if possible, it does not necessarily provide a minimal solution.

An example of this non-minimality is when the algorithm is applied to the system seen in Figure 4.19. The algorithm will use the counting function as the opacity weighting function and attempt to enforce a degree of opacity of two.

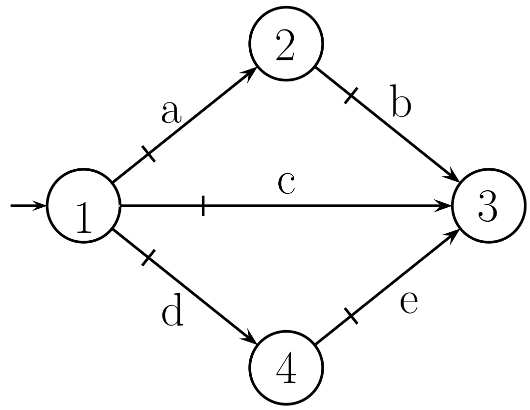
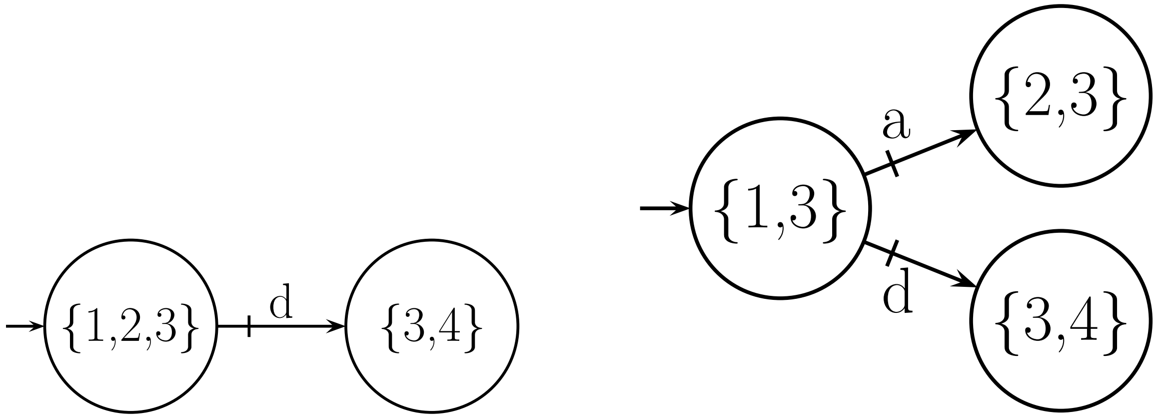


Figure 4.19: System used for disproving non-minimality of Algorithm 2

The system generated by the algorithm is presented in Figure 4.20a. This system clearly reaches the target degree of opacity. On the surface this appears to be a good result, however this is not a minimal solution. Consider the system in Figure 4.20b. While the Algorithm 2 concealed all transitions other than the transition labelled d, the system in Figure 4.20b is the observer of the system where both transition d and transition a are left observable. This system also reaches the target degree of opacity of two but only conceals a subset of the transitions that the algorithm chose to conceal. This implies that this algorithm is not guaranteed to produce a minimal solution.

One suspected reason why Algorithm 2 does not produce a minimal solution is the lack of monotonicity when transitions are concealed. Consider the example presented in Figure 4.21. Figure 4.21a shows the base system while Figure 4.21b depicts its observer. Suppose we wish to additionally conceal transition $< 2, b, 3 >$. What happens to the estimate of the system? The updated system and observer can be

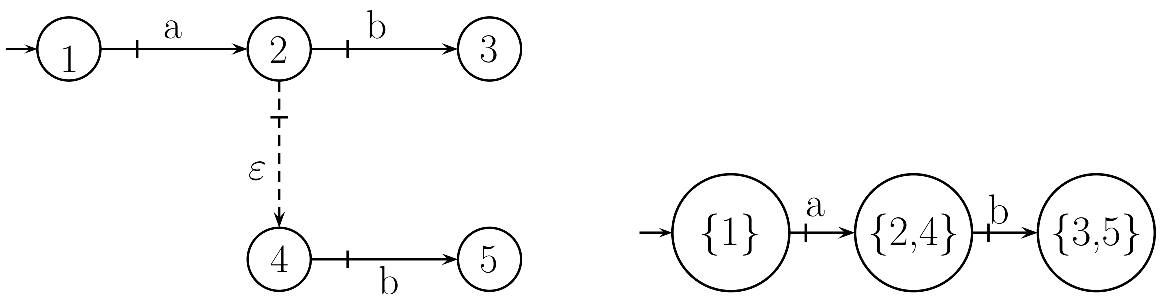


(a) Resultant system after applying Algorithm 2 to Figure 4.19

(b) Observer of the system in Figure 4.19 when only transitions a and d are kept observable

Figure 4.20: Example depicting the non-minimality of Algorithm 2

seen in Figure 4.22.

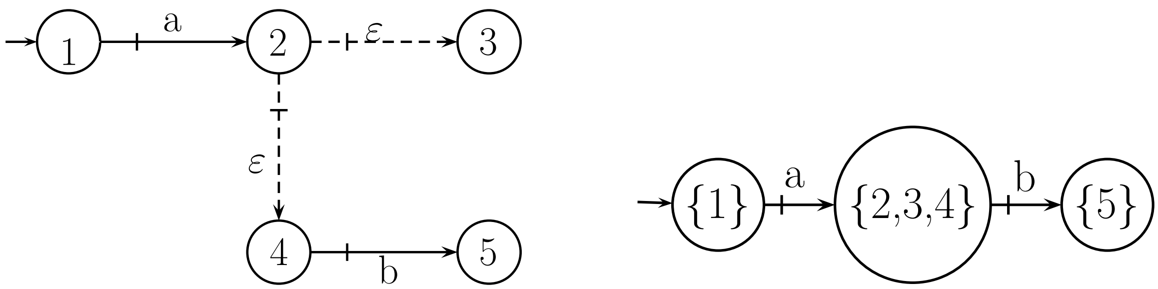


(a) Example System

(b) Observer for system in 4.21a

Figure 4.21: Base system and observer to demonstrate non-monotonicity of observability

In Figure 4.22b it can be seen that the observation with State 5 has decreased in size when compared to its estimate in the base observer. This demonstrates that the size of estimates is non-monotonic with respect to a greater number of concealed transitions. This property is present when transitions are being disabled in supervisory control and it allows for Algorithm 1 to enforce degree of opacity minimally.



(a) System after $< 2, b, 3 >$ is concealed (b) Observer for system in 4.22a

Figure 4.22: Result of concealing $< 2, b, 3 >$ for system in Figure 4.21a

The implication of non-monotonicity is that you cannot work towards a target degree of opacity by iteratively concealing and evaluating since a certain combination of concealed transitions may reduce the degree of opacity instead of adding to it.

To combat the lack of monotonicity, I developed a technique called *transition pairing*. Two transitions are paired if there exists an observation where they correspond to the same observable event in that observation. These pairs are stored in the *set of paired transitions*, which is mathematically defined below in Definition 16.

Definition (Set of Paired Transitions)**.** *The set of paired transitions, Λ, is the set of pairs of transitions where each transition in the pair corresponds to the same observable event in two different traces given some set of observable transitions T_o.*

$$\Lambda(T_o) = \{\{t_j, t'_j\} | \ \exists\ p, p' \in trace(G) \ such\ that\ P_\Sigma(P_{T_o}(p)) = P_\Sigma(P_{T_o}(p')) = \sigma_1\sigma_2...\sigma_n,$$
$$t_j =< q_j, \sigma_j, q_{j+1} > \in p, t'_j =< q'_j, \sigma_j, q'_{j+1} > \in p'\}$$

This definition does mean that it is possible for a transition to be paired with itself, however these cases do not provide any information about the system structure so they can be ignored. Also, this set is composed of set pairs. What this means is that the transitions in the set pair are not ordered, if t_i and t_j are paired, then it is sufficient to have $\{t_i, t_j\} \in \Lambda(T_o)$, you do not also include $\{t_j, t_i\}$.

The process of how pairs are formed can easily be shown with an example. Consider Figure 4.23. In this system there are three different observations, a, b, and ba. The observation a has three traces which correspond to it: $< 1, a, 3 >$, $< 1, a, 3 >< 3, \epsilon, 6 >$, and $< 1, \epsilon, 2 >< 2, a, 5 >$. Unobservable transitions are not paired, so check the transitions in each trace that correspond to the only observable event in this observation, a. The transitions that correspond to this event are $< 1, a, 3 >$, $< 1, a, 3 >$, and $< 2, a, 5 >$. Add to the set of paired transitions all available pairs of the above transitions. This means that $\{< 1, a, 3 >, < 1, a, 3 >\}$ and $\{< 1, a, 3 >, < 2, a, 5 >\}$ are in the set of paired transitions. This same process is done for all remaining observations. After removing all "self-pairs" for simplicity, the set of paired transitions is: $\Lambda(T_o) = \{\{< 1, a, 3 >, < 2, a, 5 >\}, \{< 2, a, 5 >, < 4, a, 7 >\}\}$.

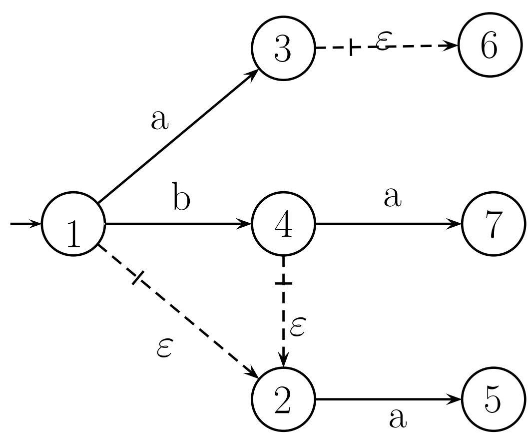


Figure 4.23: System demonstrating transition pairing

Transitions that are paired cause non-monotonic behaviour when they are concealed because they cause states to be "indistinguishable by path" instead of "indistinguishable by ϵ-reach". States that are "indistinguishable by path" are indistinguishable because the observation of the paths leading to both states look the same, but the transitions that form the observation are not the same for each trace. In this case, when an observable transition is concealed, if it was paired, the traces will no longer look the same, making the states they lead to distinguishable from each other. Since this structure now tracks which transitions are paired, when a transition is concealed, the transitions that are paired to that transition should also be concealed. The goal of this technique is have the observations of traces whose observations are the same change in the same way so no states become distinguishable if they were indistinguishable.

While this is the goal, removing just a transition and its pairs is not enough to guarantee monotonicity. Consider the system in Figure 4.23. The observer for this system is shown in Figure 4.24.

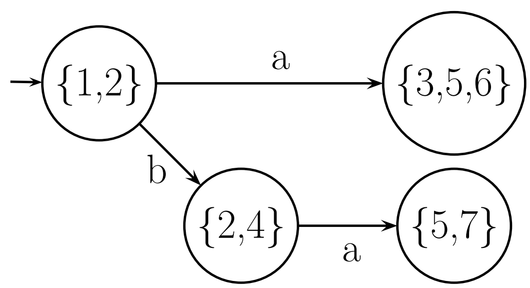


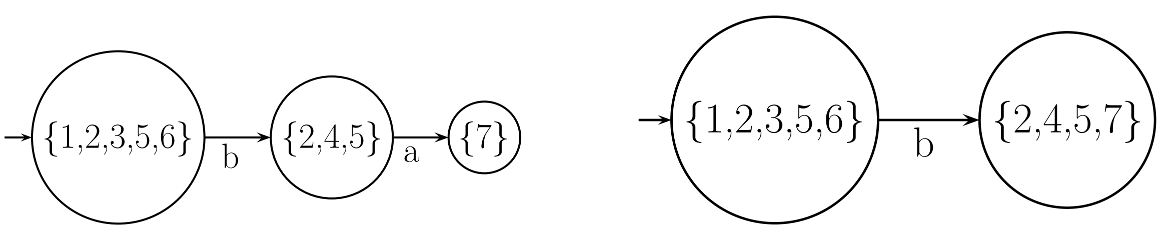
Figure 4.24: Observer for system in Figure 4.23

Say we wish to conceal $< 1, a, 3 >$. If we only conceal it and its pairs, we would conceal $< 1, a, 3 >$ and $< 2, a, 5 >$. This would lead to the updated observer shown in Figure 4.25a. In this case, the estimate with State 7 has decreased in size from its counterpart estimate with State 7 in the original observer. Since the size of this state goes down, concealing a transition and its pairs will not guarantee monotonicity of the size of estimates. For this specific example, this occurs because $< 2, a, 5 >$ is paired to $< 1, a, 3 >$ and $< 4, a, 7 >$ but $< 1, a, 3 >$ is not paired with $< 4, a, 7 >$.

This intuitively makes sense, since we conceal $< 1, a, 3 >$'s pair in order to preserve the indistinguishability of traces, we will need to conceal the pairs of all transitions we conceal to preserve indistinguishability. This means that we now need to remove the pairs of $< 2, a, 5 >$, which would include $< 4, a, 7 >$ and any pairs that $< 4, a, 7 >$ had and so on. Mathematically, we are concealing the transitive closure of the transition $< 1, a, 3 >$. This transitive closure is explicitly defined as follows.

Definition (Transitive Closure of Paired Transitions)**.** *The transitive closure of paired transitions is $trcl_{\Lambda(T_o)}$ where $trcl$ is shorthand for transitive closure. This function, when given a transition t, returns the set of all transitions that are in its transitive closure within the set of paired transitions $\Lambda(T_o)$.*

Apply the transitive closure technique to the above example. As stated above, this would conceal $< 4, a, 7 >$ in addition to $< 1, a, 3 >$ and $< 2, a, 5 >$ which were concealed before. The new observer, after all three of these have been concealed, is presented in Figure 4.25b. In this observer, State 7's estimate has been combined with another and increases in size, no longer decreasing.



(a) Example System

(b) Observer for system in 4.5a

Figure 4.25: Comparison of concealing just a transition's pairs versus its transitive closure

The above example showed that concealing the transitive closure of a transition leads to no estimate size decreases for that system. Proposition 1 and its proof shows that this is true for all cases.

Proposition 1. *Consider a system* $G = (Q, \Sigma, q_0, \delta_G, Q_m)$ *with set* T_o. *Conceal* $trcl_{\Lambda(T_o)}(t)$ *where* $t \in T_o$. *Let* $T'_o = T_o \backslash trcl_{\Lambda(T_o)}(t)$. *Consider* $s \in trace(G)$. *Let* $p = P_\Sigma(P_{T_o}(s))$ *and* $p' = P_\Sigma(P_{T'_o}(s))$. *It is the case that* $tr(p, T_o) \subseteq tr(p', T'_o)$.

Proof. Proceed by contradiction. Suppose that $tr(p, T_o) \not\subseteq tr(p', T'_o)$.

This means that $\exists w \in tr(p, T_o)$ where $w \notin tr(p', T'_o)$. Specifically, this implies that trace w has the same observation as s under observable transition set T_o but not under the observable transition set T'_o. Let $p'' = P_\Sigma(P_{T'_o}(w))$ be the observation for w under T'_o (and moreover, $p' \neq p''$).

Look at the strings of observable transitions under T_o. Let $P_{T_o}(s) = t_{s,0}t_{s,1}...t_{s,n}$ and $P_{T_o}(w) = t_{w,0}t_{w,1}...t_{w,n}$ for $t_{s,i}, t_{w,i} \in T_o$, $i = 0, 1, ..., n$ and $n \in \mathbb{Z}_{\geq 0}$. Since $T'_o \subseteq T_o$ and $p' \neq p''$, $\exists\ j \in \{0, 1, ..., n\}$ such that ($t_{s,j} \in T'_o$ and $t_{w,j} \notin T'_o$) or ($t_{s,j} \notin T'_o$ and $t_{w,j} \in T'_o$). One of these must be the case for the observations of the strings to be different under the new observable transition set, T'_o.

Since $t_{s,i}, t_{w,i} \in T_o$ for any $i = 0, 1, ..., n$ and $T'_o = T_o \backslash trcl_{\Lambda(T_o)}(t)$ then if a given $t_{s,j}$ or $t_{w,j}$ is not in T'_o, it must be in $trcl_{\Lambda(T_o)}(t)$. Now consider the case of ($t_{s,j} \in T'_o$ and $t_{w,j} \notin T'_o$). Then $t_{w,j} \in trcl_{\Lambda(T_o)}(t)$. Since $s, w \in tr(p, T_o)$, this by definition means that both s and w have the same observation under T_o, which in turn means $\{t_{s,i}, t_{w,i}\} \in \Lambda(T_o) \forall i = 0, 1, ..., n$. This means that if $t_{w,j} \in trcl_{\Lambda(T_o)}(t)$ then $t_{s,j} \in trcl_{\Lambda(T_o)}(t)$ and hence $t_{s,j} \notin T'_o$ which contradicts the assumption that $t_{s,j} \in T'_o$. The case of ($t_{s,j} \notin T'_o$ and $t_{w,j} \in T'_o$) can be proven analogously. □

The nature of this proof can be demonstrated with a small example. Consider the system in Figure 4.26. In this system, State 4 can be reached by the observation *aba*. The broken arrow represents a trace s that leads to State 5. All that is known about trace s is that its observation is aba ($P_\Sigma(P_{T_o}(s)) = aba$). What happens if we choose to conceal the transition $< 1, a, 2 >$, the transition corresponding to the first a event on the path to State 4, and its transitive closure? If we do this, suppose the observation of the trace leading to State 4 becomes ba. What happens to the observation of the trace s which leads to State 5? Since we do not explicitly know the transitions which comprise s, it is not obvious what its resultant observation will be. Recall that Proposition 1 hopes to establish that all traces with the same observation will still share an observation after any number of transitions are concealed if they are concealed with the pairing method. In this example, that would mean that all traces that look like aba will look like after ba after the transitions are concealed if it is known that one such trace's resultant observation is ba. To show why this has to be the case, suppose that the resultant observation of s is *not* ba. There are two possible cases of what the observation would look like instead:

(i) the observation to State 5 has kept an event that the observation to State 4 has not

(ii) the observation to State 5 has removed an event that the observation State 4 has not

These cases can be combined to capture any way in which these observations could be different. For this example, in case (i) the observation for State 5 may stay as aba while for the case (ii) it may have the observation look like just b. Are these options possible given our problem construction? It is important to consider how we are concealing the transitions; we are concealing the transitive closure of the transitions based on their pairing. Since the traces to State 4 and State 5 have the same observation initially, the transitions corresponding to each observable event will be paired by definition. For case (i), since the transitions corresponding to the first a event are paired, if the transition on the trace to State 4 is concealed, its pair, the transition on the trace to State 5, must also be concealed. This means that case (i) is not possible. A similar thing occurs in case (ii). Since the second a in the trace to State 4 would have been concealed through pairing if the second a in the trace to State 5 was, we know that case (ii) is also not possible. Since neither of these cases are possible, by extension, any combination will be impossible also. This means that the observation leading to State 5 must now be ba as well. Since the observations of these traces change in the same way, no indistinguishable states are split, so concealing events is monotonic.

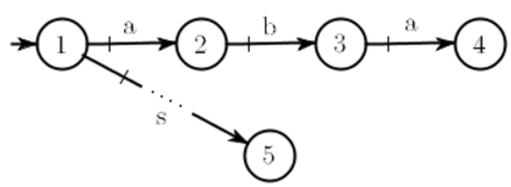


Figure 4.26: Example to demonstrate the proof of Proposition 1

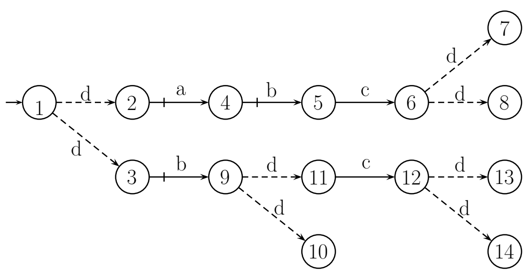
Proposition 1 can now be applied in an algorithm to enforce degree of opacity by concealing events. Algorithm 3 is based off of Algorithm 1 which uses supervisory control to enforce DoO but instead chooses transitions to conceal.

Algorithm 3 does indeed enforce the target degree of opacity by concealing events. Unfortunately, even through the use of the paired transition technique, Algorithm 3 does not necessarily produce a minimal solution. This is shown by applying the algorithm to the example system in Figure 4.27a. Its corresponding observer is seen in Figure 4.27b. Use the counting function as the opacity weighting function and set the target degree of opacity to three.

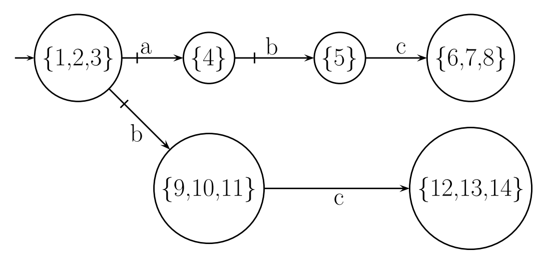
Data: System $G = (Q, \Sigma, q_0, \delta_G)$, the adversary's observable transitions, $T_o \subseteq T$, and a degree of opacity, μ.
Result: A supervisor S such that $\Theta_{T_o}(S/G) \geq \mu$, or notification that no solution exists.

Compute $H = \text{Obs}_{T_o}(G)$;
Compute $M = G \mid\mid H$;
Mark all states $x = \frac{q}{A}$ in M where $w(A) < \mu$;
if $w(Q) < \mu$ **then**
  return No solution exists ;
Set Δ as an empty list ;
while *M has marked states* **do**
  foreach *marked state $x = \frac{q}{A}$ in M* **do**
    foreach *transition t leading into x, where $t =< x', \sigma, x >$ and $x' = \frac{q'}{A'}$* **do**
      Find the corresponding transition in G, $t' =< q', \sigma, q >$;
      Add $trcl_{\Lambda(T_o)}(t')$ to Δ;
  Compute $H = \text{Obs}_{T_o \setminus \Delta}(G)$;
  $M = G \mid\mid H$;
  Mark all states $x = \frac{q}{A}$ in M where $w(A) < \mu$;
return $S = M$;

Algorithm 3: Supervisor synthesis for Problem 3 by concealing events



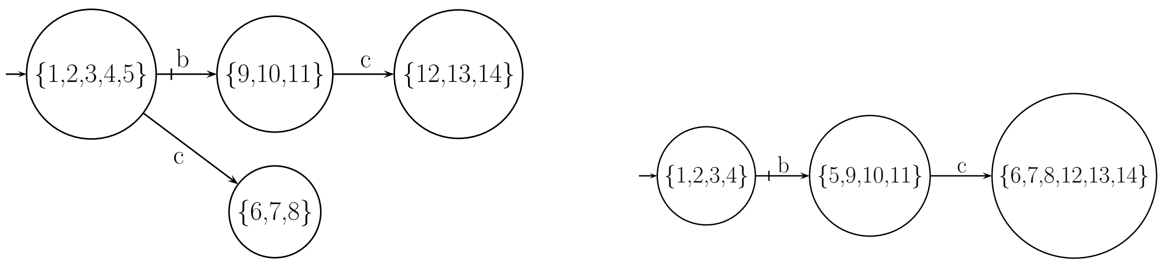
(a) System used for disproving non-minimality of Algorithm 3. Events a, b, c are observable and event d is concealed



(b) Observer for the system in Figure 4.27a

Figure 4.27: Systems to be used to show the non-minimality of Algorithm 3

The algorithm chooses to conceal transitions $< 2, a, 4 >$ and $< 4, b, 5 >$ in the plant since they lead to states whose estimates violate the degree of opacity. The recalculated observer is shown in Figure 4.28a and it does indeed enforce the target degree of opacity of three. However, if you choose to only conceal the transition $< 2, a, 4 >$ in the plant, then the target degree of opacity is still enforced, as can be seen in Figure 4.28b. This shows that this updated algorithm is also not minimal, meaning that pairing transitions is not enough to guarantee minimality.



(a) Resultant system after applying Algorithm 3 to Figure 4.27b

(b) Observer of the system in Figure 4.27b when only $< 2, a, 4 >$ is concealed

Figure 4.28: Example depicting the non-minimality of Algorithm 3

The reason why this iteration of the algorithm did not provide a minimal solution is again tied to “indistinguishability by path”. In the above example, it suffices to conceal only $< 2, a, 4 >$ because doing so causes State 5 to be indistinguishable by path with States 9, 10, and 11. Algorithm 3 works like the degree of opacity enforcing algorithm which uses supervisory control, Algorithm 1. If a state is marked, the algorithm chooses to conceal the transitive closure of the transitions leading to that state. If states could not be indistinguishable by path (e.g., if all transition labels were unique), this would be a minimal solution. However with path indistinguishability, there could be a way to obscure a marked state with fewer concealed transitions if the algorithm targets transitions that will cause the observation of the marked state's trace to be indistinguishable with an observation that leads to an estimate which reaches the target weighting.

Chapter 5

Conclusion

5.1 Conclusion

Participation with the industry project led me to apply DES to a new class of situations. In particular, I focused on autonomous vehicles that traverse terrain to accomplish a given task, such as locating a target. When multiple vehicles work together to accomplish a task, how can it be done efficiently while also taking steps to protect the agents' and target's locations? If there is only a pair of vehicles searching terrain that has no loops, we were able to develop a priority movement policy that would enable the search to be fully efficient while eliminating the need for any communication [10]. If the above conditions are not met and communication is unavoidable, then how can a certain level of security be computed and enforced? This problem was addressed by introducing the concept of degree of opacity. Traditional opacity has restrictions which makes it impossible to apply to the terrain search situation. Degree of opacity extends traditional opacity to be incremental and incorporates properties that are important to the system to calculate the level of security. This was applied to the terrain search problem and enforced by either restricting the vehicles' movement or communications.

This paper introduced a variety of new strategies to handle security in a terrain search situation. While communication can pose large threats to many systems, the priority movement policy demonstrates that in specific scenarios communication can be eliminated with a bit of ingenuity. In the same vein, traditional opacity was used as inspiration to create a new opacity definition. Since traditional opacity is a binary notion, opaque or transparent, and is based in a specific set of secrets, it makes it difficult to apply to some problems, like the terrain search scenario. Degree of opacity, introduced in this paper, eliminates both of these issues while maintaining all of the properties of traditional opacity, making it a more general framework. It utilizes an arbitrary weighting function which allows for degree of opacity to be customized to capture important security properties in a given system. With a proper choice of weighting function, you can calculate a degree of opacity for a given observation, making opacity a metric instead of a binary concept. System designers can then select what security tolerance level is appropriate for their system and adjust the system to reach this target security level. Since each estimation can be evaluated separately, an overall degree of opacity can be calculated for the system, regardless of whether there is a set of secret states in the system. In the same way that it is useful to be able to enforce traditional notions of opacity on a system by manipulating system events, it is useful to be able to enforce degree of opacity through event manipulation as well. This paper showed that both controllability and observability of events can be leveraged to enforce degree of opacity. Through supervisory control, degree of opacity enforcement was proven to be minimal. Leveraging observability presented a larger issue where minimality is concerned. Concealment of transitions was found to be non-monotonic, a property that is inherent in controllability. A transition pairing technique was introduced that was proven to make the concealment of transitions monotonic. Unfortunately, monotonicity alone was not enough to guarantee the minimality of a degree of opacity enforcement through concealment algorithm.

The priority movement policy is only an effective strategy in certain situations. It only produces a maximally efficient search if the terrain does not have any loops and if there are only two vehicles participating in the search. Work into how this could be extended to systems without loops was done in Kulchyk's thesis [11]. Since priority movement policies rely on reverse policies to produce a maximally efficient search, without knowledge of the system beforehand, two is the maximum number of vehicles that can be used to produce a maximally efficient search using reverse priority policies. A probabilistic analysis was performed in Chapter 3 to present good policy candidates for additional vehicles, but the effect on the search efficiency was not demonstrated since it would depend heavily on the system being searched.

Degree of opacity is a more general framework than traditional opacity. It has the same limitations as opacity, except for its binary nature and its dependence on a set of secrets. The weighting function used to calculate the degree of opacity does have some limitations. This function must be non-negative and must be monotonic with respect to subset inclusion. These restrictions reflect properties that are normally seen when considering opacity, however it is possible to develop a situation that does not adhere to these restrictions. Enforcing degree of opacity through supervisory control was shown to be minimal with an algorithm that is not exponential, but using only event controllability can be restrictive, potentially causing the system goal, like finding a target in a terrain search, to be impossible to achieve. On the other hand, enforcing degree of opacity through concealing events does not restrict accessibility to any states in the system. However properties of observability make it challenging to enforce minimally and this work was not able to develop a strategy that would guarantee a minimal solution.

In conclusion, degree of opacity provides a general nonbinary metric to measure how opaque a system is and consequently allows overall system security or the security of a secret to be quantified.

5.2 Future Work

In the future, I hope this framework is used as a tool to analyze the security of different systems in a quantifiable way. This research would benefit from more work done in the area of enforcement by concealing transitions. Work to continue to develop a minimal enforcement strategy would be extremely useful. At the very least, it would be beneficial to see if the need to recalculate observers could be removed so that it could be enforced in polynomial time. Alternatively, it would be useful for the current algorithm if it could be proven to be at greatest a certain amount off of being minimal.

Bibliography

[1] W. M. Wonham and K. Cai, *Supervisory Control of Discrete-Event Systems*, ser. Communications and Control Engineering. Springer International Publishing, 2019.

[2] C. G. Cassandras and S. Lafortune, *Introduction to Discrete Event Systems*, 2nd ed. Boston, MA: Springer US, 2008.

[3] P. J. Ramadge and W. M. Wonham, "Supervisory Control of a Class of Discrete Event Processes," *SIAM Journal on Control and Optimization*, vol. 25, no. 1, pp. 206–230, Jan 1987.

[4] A. Saboori and C. N. Hadjicostis, "Notions of Security and Opacity in Discrete Event Systems," *Proceedings of the IEEE Conference on Decision and Control*, pp. 5056–5061, 2007.

[5] F. Lin, "Opacity of Discrete Event Systems and Its Applications," *Automatica*, vol. 47, no. 3, pp. 496–503, Mar 2011.

[6] A. Saboori and C. N. Hadjicostis, "Opacity Verification in Stochastic Discrete Event Systems," in *49th IEEE Conference on Decision and Control (CDC)*. IEEE, Dec 2010, pp. 6759–6764.

[7] Y. C. Wu and S. Lafortune, "Comparative Analysis of Related Notions of Opacity in Centralized and Coordinated Architectures," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 23, no. 3, pp. 307–339, 2013.

[8] J. Košeckà and R. Bajcsy, "Discrete event systems for autonomous mobile agents," *Robotics and Autonomous Systems*, vol. 12, no. 3, pp. 187–198, 1994. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0921889094900256

[9] J. Kosecka and L. Bogoni, "Application of discrete events systems for modeling and controlling robotic agents," *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pp. 2557–2562 vol.3, 1994.

[10] J. Kulchyk, B. Schonewille, R. Ean, and K. Rudie, "Communication-free multi-agent coordination in an unknown environment," *Proceedings of the 15th IFAC Workshop on Discrete Event Systems WODES 2020*, pp. 159–165, 2020.

[11] J. R. Kulchyk, "Security by design: Reducing information exchange in multi-agent search tasks," Master's thesis, Department of Electrical and Computer Engineering, Queen's University, 2021.

[12] B. Bérard, J. Mullins, and M. Sassolas, "Quantifying Opacity," *Mathematical Structures in Computer Science*, vol. 25, no. 2, pp. 361–403, Feb 2015.

[13] A. Saboori and C. N. Hadjicostis, "Current-State Opacity Formulations in Probabilistic Finite Automata," *IEEE Transactions on Automatic Control*, vol. 59, no. 1, pp. 120–133, Jan 2014.

[14] B. Schonewille, R. Moulton, and K. Rudie, "Enforcing degree of opacity with supervisory control," *Submitted to American Control Conference*, 2022.

[15] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, 1958.

[16] L. Ford, *Network Flow Theory.* Rand Corporation, 1956.

[17] R. Moulton, B. Behinaein Hamgini, Z. Abedi Khouzani, R. Meira-Góes, F. Wang, and K. Rudie, "Using subobservers to synthesize opacity-enforcing supervisors," *Manuscript submitted for publication*, 2021.

[18] J. Dubreil, P. Darondeau, and H. Marchand, "Supervisory Control for Opacity," *IEEE Transactions on Automatic Control*, vol. 55, no. 5, pp. 1089–1100, 2010.

[19] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Current-State Opacity Enforcement in Discrete event systems Under Incomparable Observations," *Discrete Event Dynamic Systems: Theory and Applications*, vol. 28, no. 2, pp. 161–182, 2018.

[20] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Automata Theory, Languages, and Computation*, ser. Communications and Control Engineering. Pearson Addison Wesley, 2007.

[21] W. M. Wonham and P. J. Ramadge, "On the supremal controllable sublanguage of a given language," *SIAM Journal on Control and Optimization*, vol. 25, no. 3, pp. 637–659, 1987.

[22] J. W. Bryans, M. Koutny, L. Mazaré, and P. Y. A. Ryan, "Opacity generalised to transition systems," *International journal of information security*, vol. 7, no. 6, pp. 421–435, 2008.

[23] N. A. Lynch, *Distributed Algorithms*, ser. Morgan Kaufmann series in data management systems, 1996.

[24] K. Rudie, "The Integrated Discrete-Event Systems Tool," *8th International Workshop on Discrete Event Systems*, pp. 394–395, 2006.

Appendix A

`REFINE` Algorithm

Data: Automaton $G \mid\mid H$, Δ, a list of states to make inaccessible in $G \mid\mid H$.
Result: An automaton M, the accessible portion of $G' \mid\mid H'$.

$M = G \mid\mid H$;
`/*` $A_r(A)$ `is a set of states that should be removed from the adversary's estimate` A. `*/`
Initialize $A_r(\cdot)$ as $\emptyset$ for each estimate;
`/* Determine from` Δ `if any other states should be made inaccessible. */`
$\Delta_Q = \Delta$;
while Δ_Q *is not empty* **do**
    foreach *transistion leading into* $\frac{q}{A}$, $t = \langle \frac{q'}{A'}, \sigma, \frac{q}{A} \rangle_M$ **do**
        if $\sigma \in \Sigma_{uc}$ **then** the transistion t is uncontrollable
            Add $\frac{q'}{A'}$ to Δ_Q;
    Remove $\frac{q}{A}$ from Δ_Q;
    Remove $\frac{q}{A}$ from M;
`/* States in` M `sharing the same estimate as newly inaccessible states need to be updated. */`
foreach *State* $\frac{q}{A}$ *in* $G \mid\mid H$ **do**
    if $\frac{q}{A}$ *is not accessible in* M **then**
        $A_r(A) = A_r(A) \cup q$;
`/* Relabel states in` M `to account for the observer's updated estimates of the plant. */`

foreach *State* $\frac{q}{A}$ *in* M **do**
  Relabel $\frac{q}{A}$ to $\frac{q}{A \setminus A_r(A)}$;
`/* Ensure only the accessible portion of` M `is returned. */`
$M = Ac(M)$;

Algorithm 4: `REFINE` Algorithm